Crafting scalable AI for a AAA Souls-like game in Unreal Engine

Past, Present and Future for the AI in Lords Of The Fallen

Bruno Rebaque – Senior AI Programmer

Talk Focus

- *Navigation*

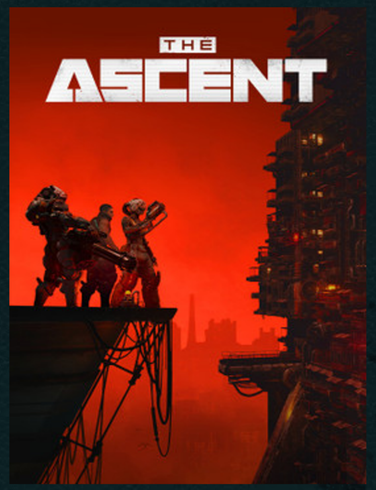
  The concept of “One Nav For All”
- *Decision Making*

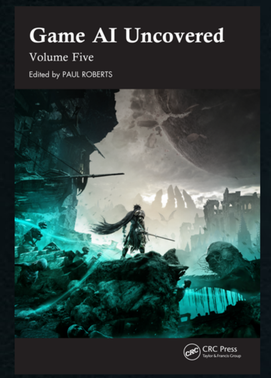
  Data Driven Approach and the evolution from monolotic BT’s to the Reaction System
- *Combat*

  Aggro and Combat Roles for AI

Who am I?

Bruno Rebaque – Senior AI Programmer





Game AI Uncovered by Paul Roberts

SUMO DIGITAL

NEON GIANT

GAMES

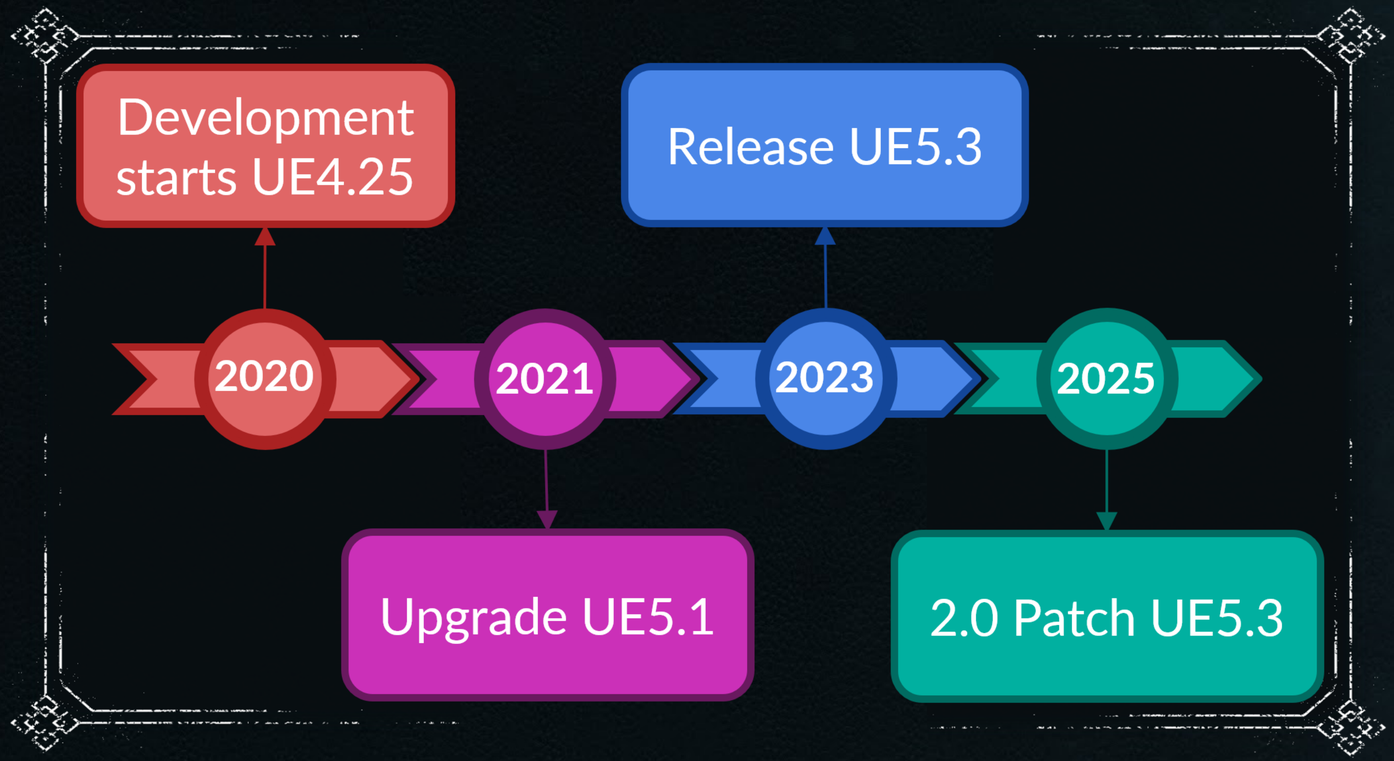
GAMES

CRC Press
Taylor & Francis Group

Lords of the Fallen

Project context

- Fast Iteration + short development time + long time support



Lords of the Fallen

Project context

- Fast Iteration + short development time + long time support
- Aimed for Xbox X/S, PS5 and Low/Mid PC's (by 2020's standards)

SYSTEM REQUIREMENTS

MINIMUM:	RECOMMENDED:
OS: Windows 10 64bit	OS: Windows 10 64bit
Processor: intel i5 8400 | AMD Ryzen 5 2600	Processor: intel i7 8700 | AMD Ryzen 5 3600
Memory: 12 GB RAM	Memory: 16 GB RAM
Graphics: 6GBs VRAM | NVIDIA GTX-1060 | AMD Radeon RX 590	Graphics: 8GBs VRAM | NVIDIA RTX-2080 | AMD Radeon RX 6700
DirectX: Version 12	DirectX: Version 12
Storage: 45 GB available space	Network: Broadband Internet connection
Additional Notes: 720p Low Quality Settings (30FPS) | SSD (Preferred) | HDD (Supported)	Storage: 45 GB available space
	Additional Notes: 1080p High Quality Settings (60FPS) | SSD Required

Lords of the Fallen

Project context

- Fast Iteration + short development time + long time support
- Aimed for Xbox X/S, PS5 and Low/Mid PC's (by 2020's standards)

Game context

- Fast paced and reactive combat with high-balanced difficulty
- Multiple maps with very different environments
- Vast variety of enemies (numbers and behaviours)

NAVIGATION

Navigation

- *CPU Limitations*

  Dynamic With Modifiers Only

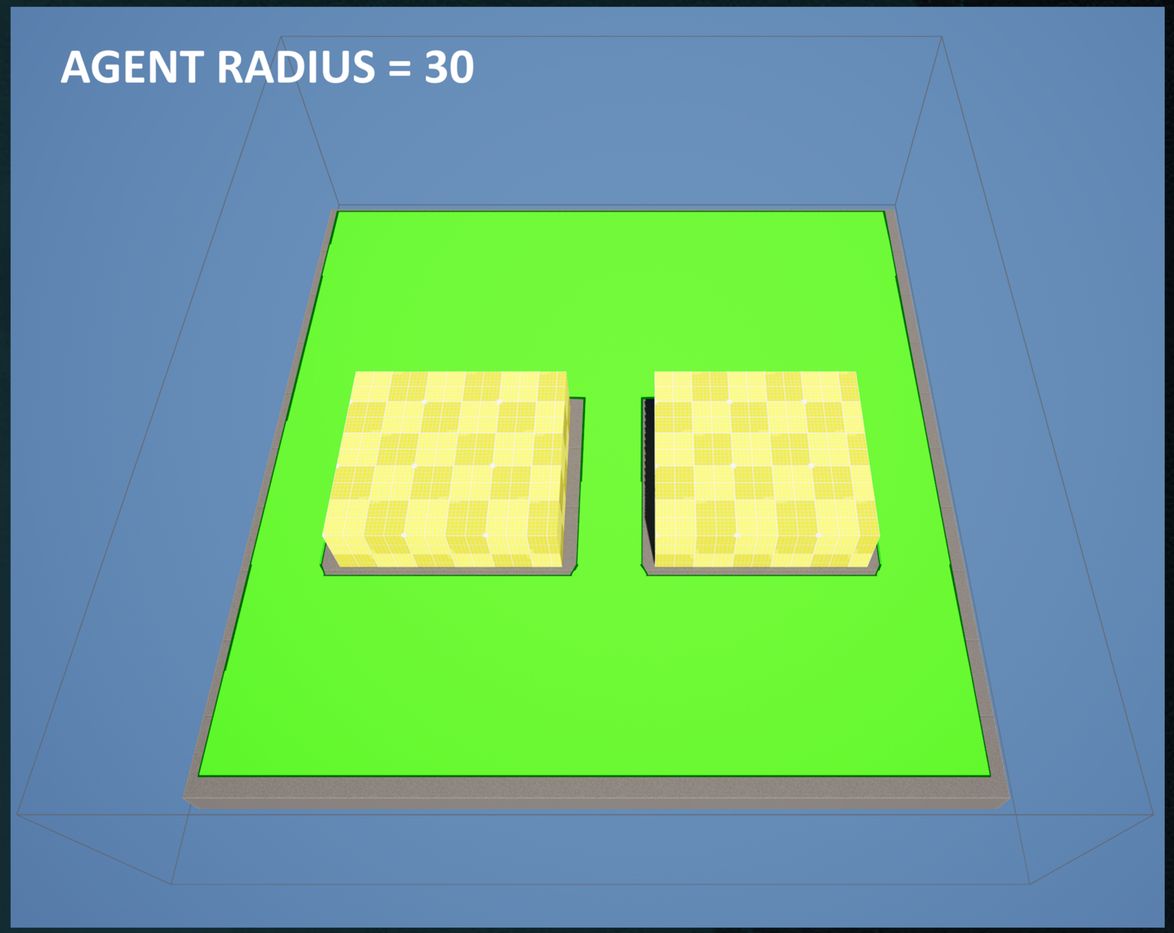
- *Memory Limitations*

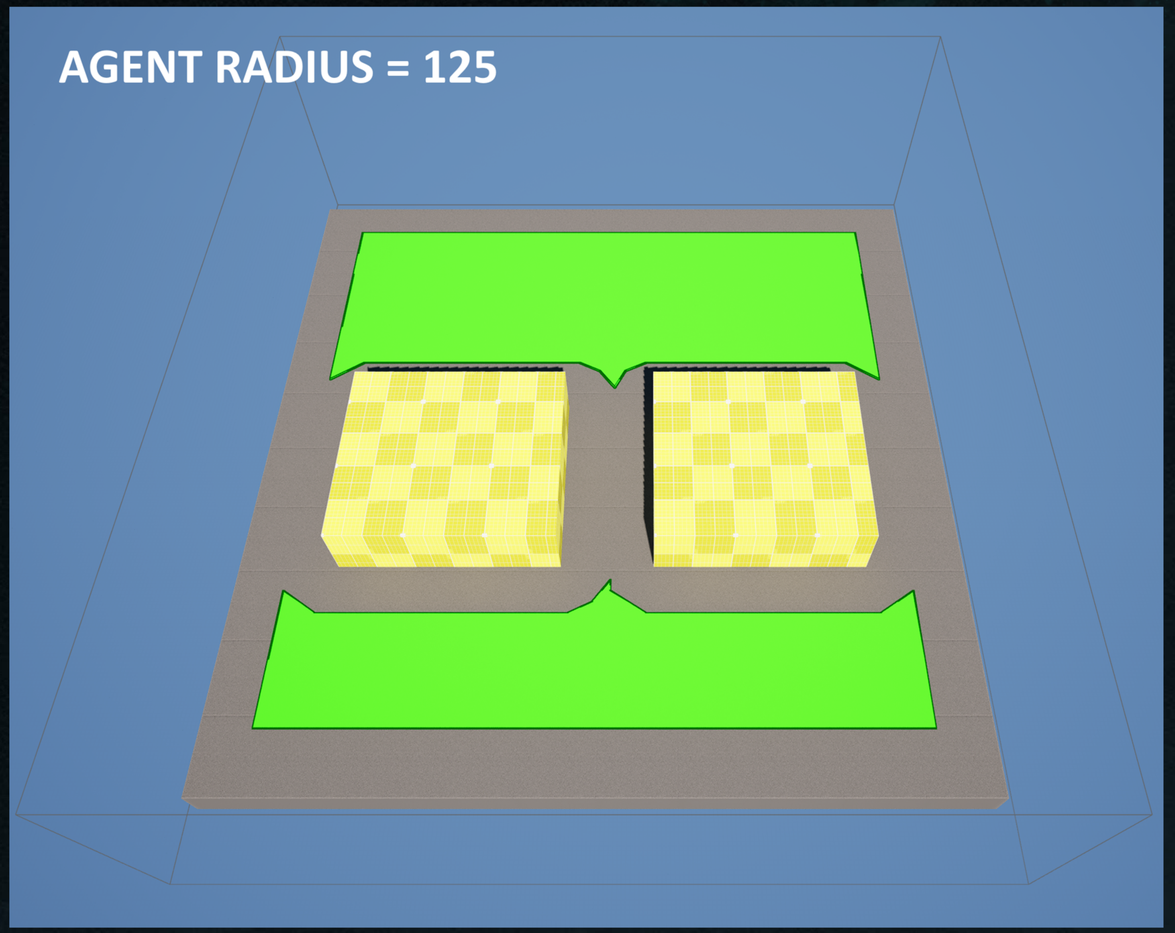
  World Composition with Nav Bound Volumes in Sub-Levels + Nav Streaming (Nav Data Chunks)

Navigation

➢ Nav data

Too many Nav Agents

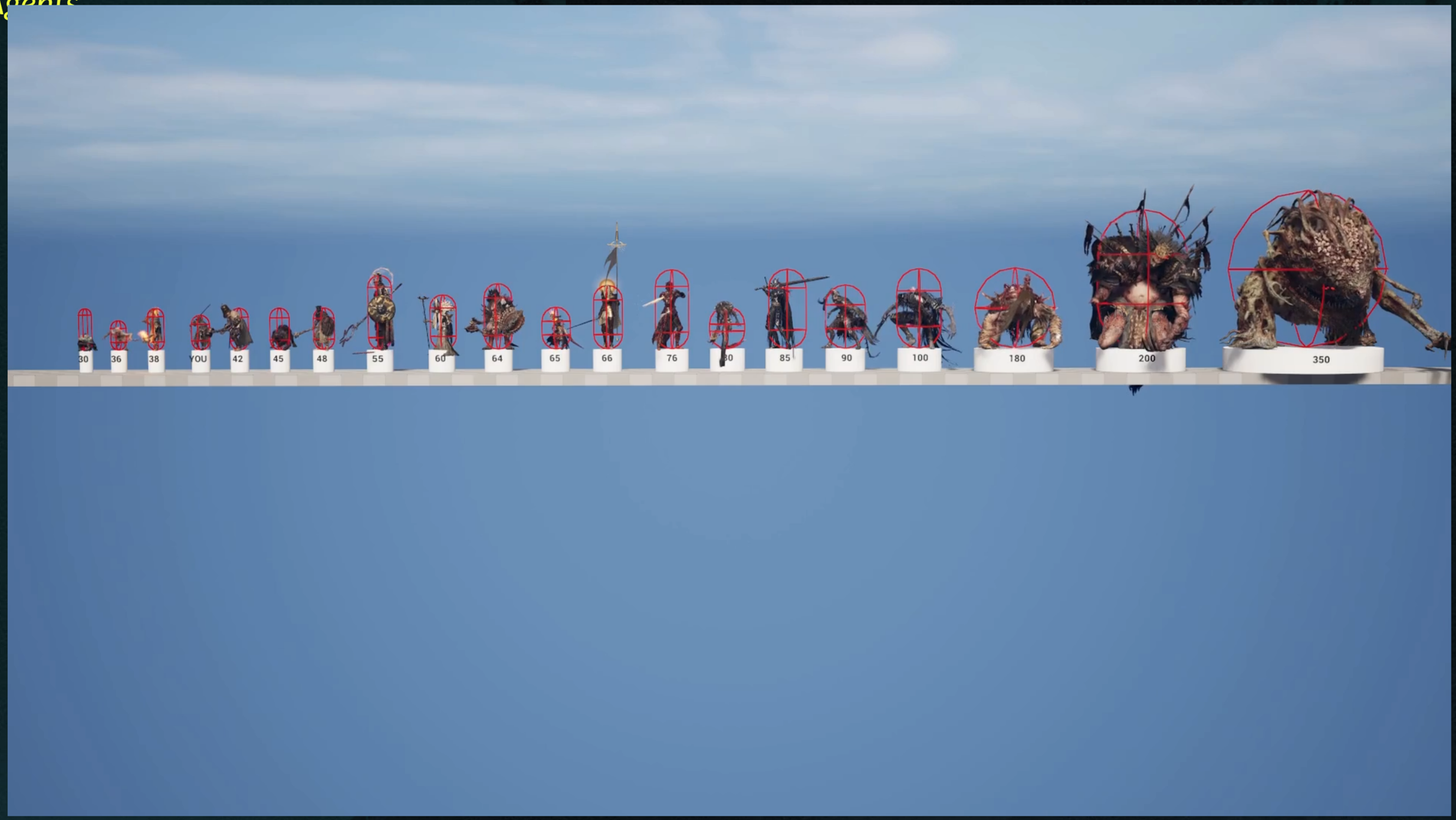




Navigation

➢ Nav data

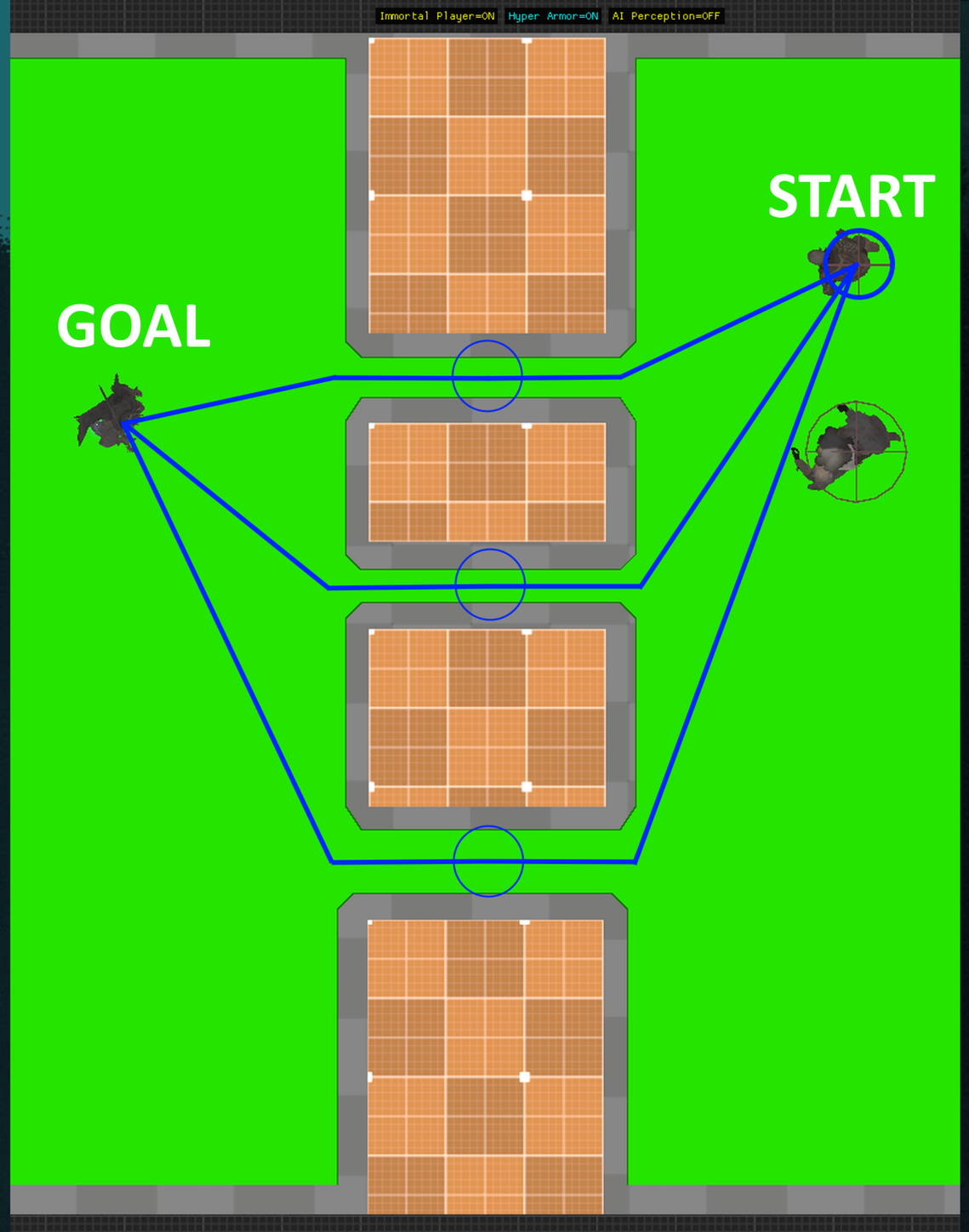
Too many Nav Agents



Navigation

➢ Nav data

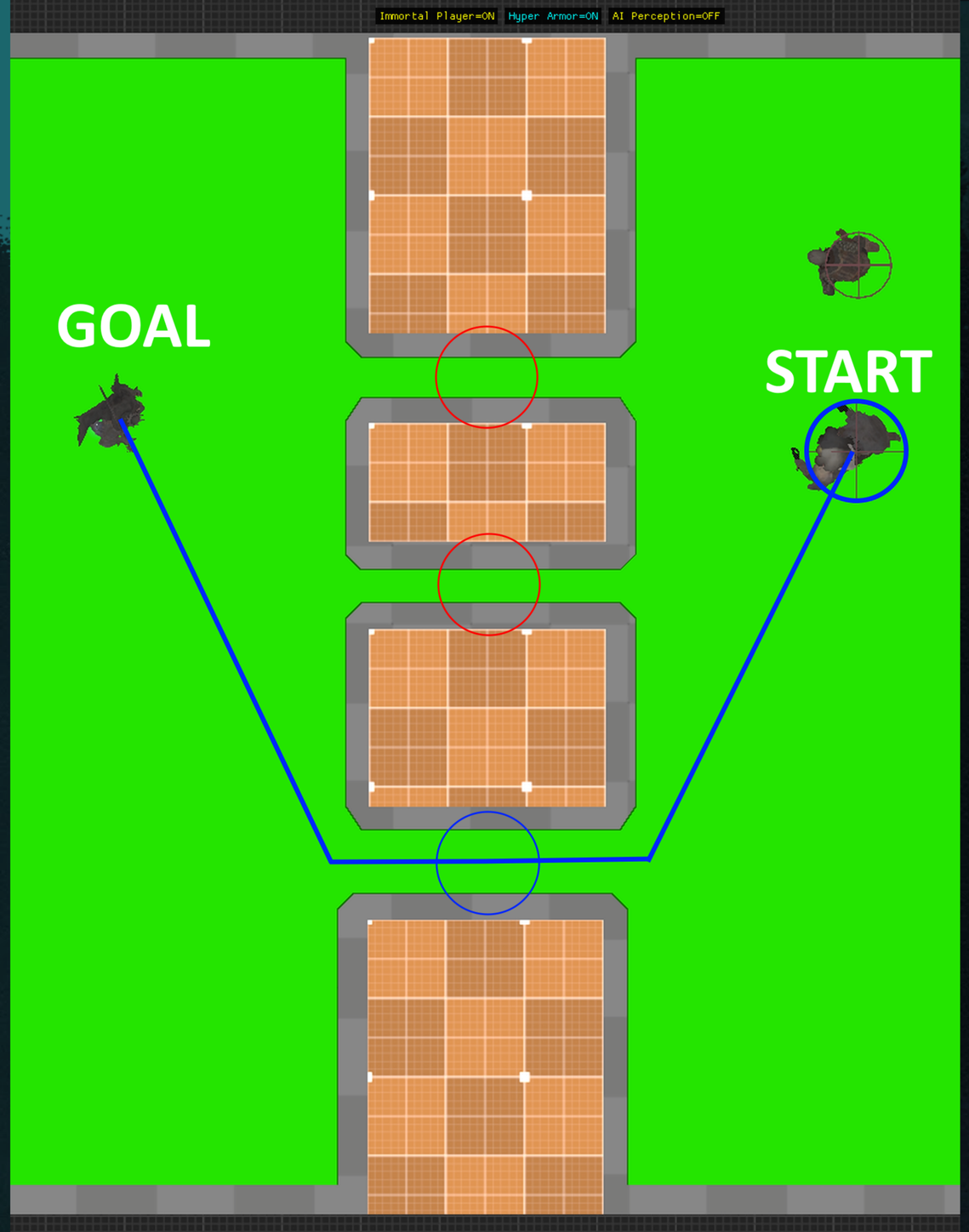
Too many Nav Agents



Navigation

➢ Nav data

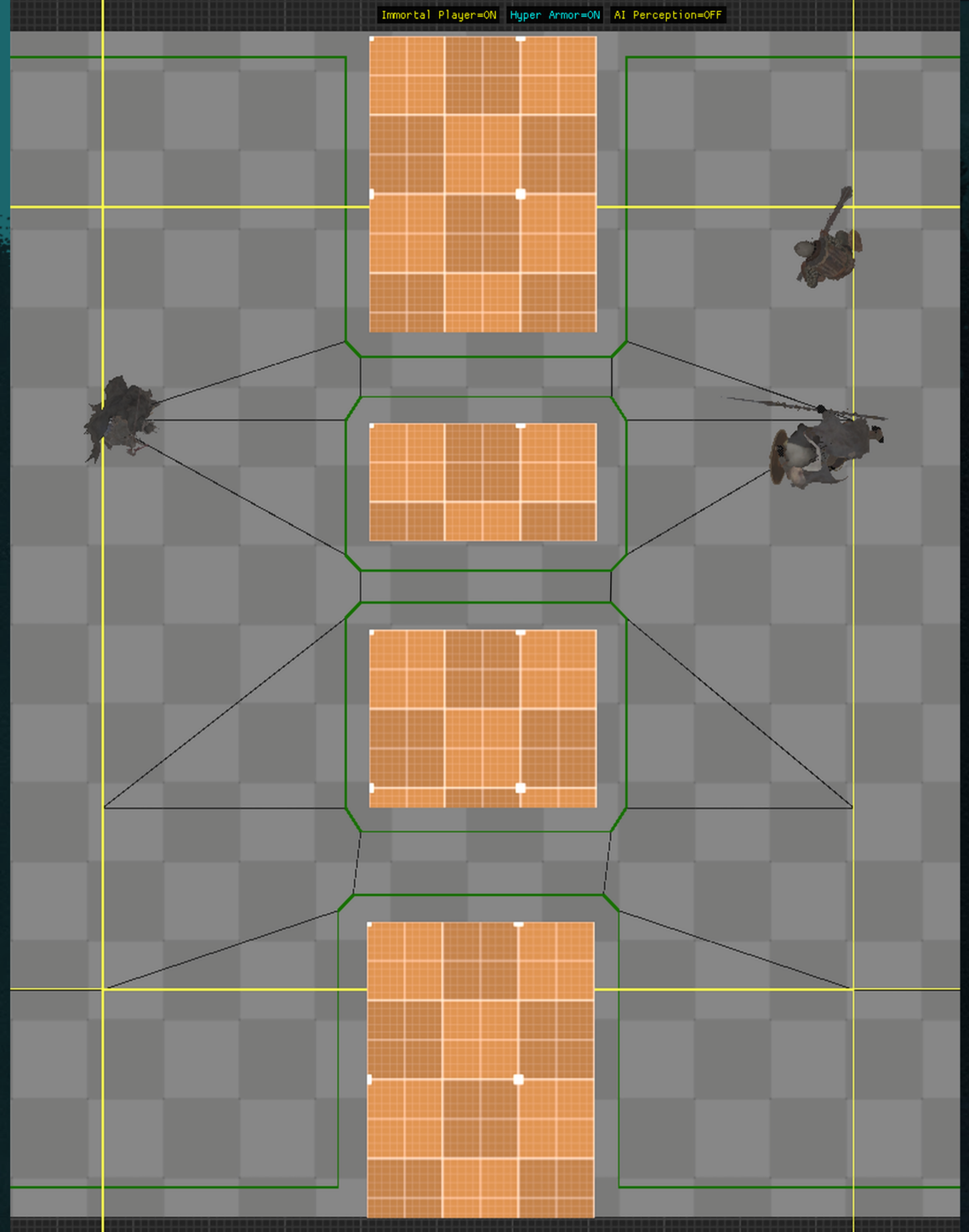
Too many Nav Agents

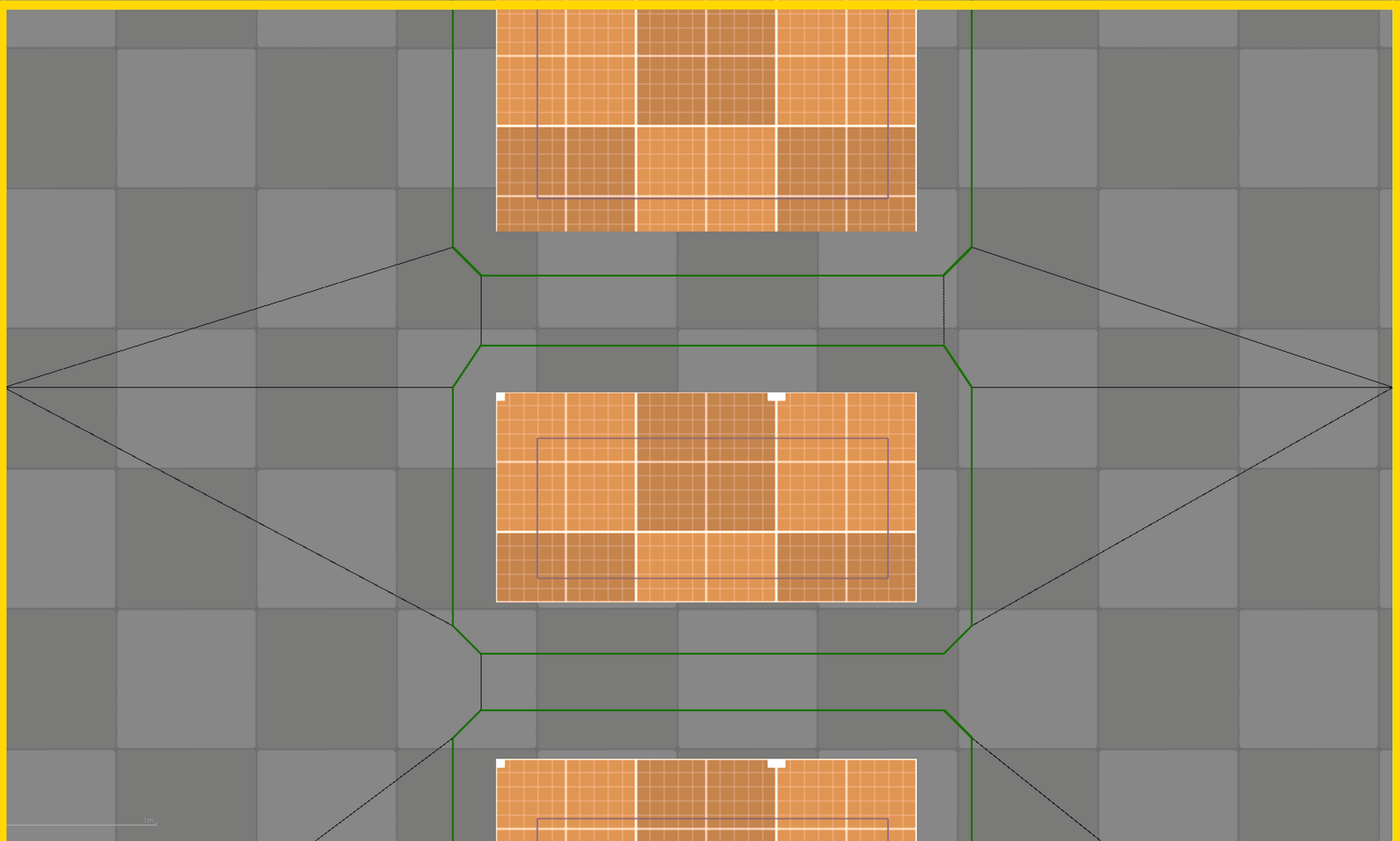


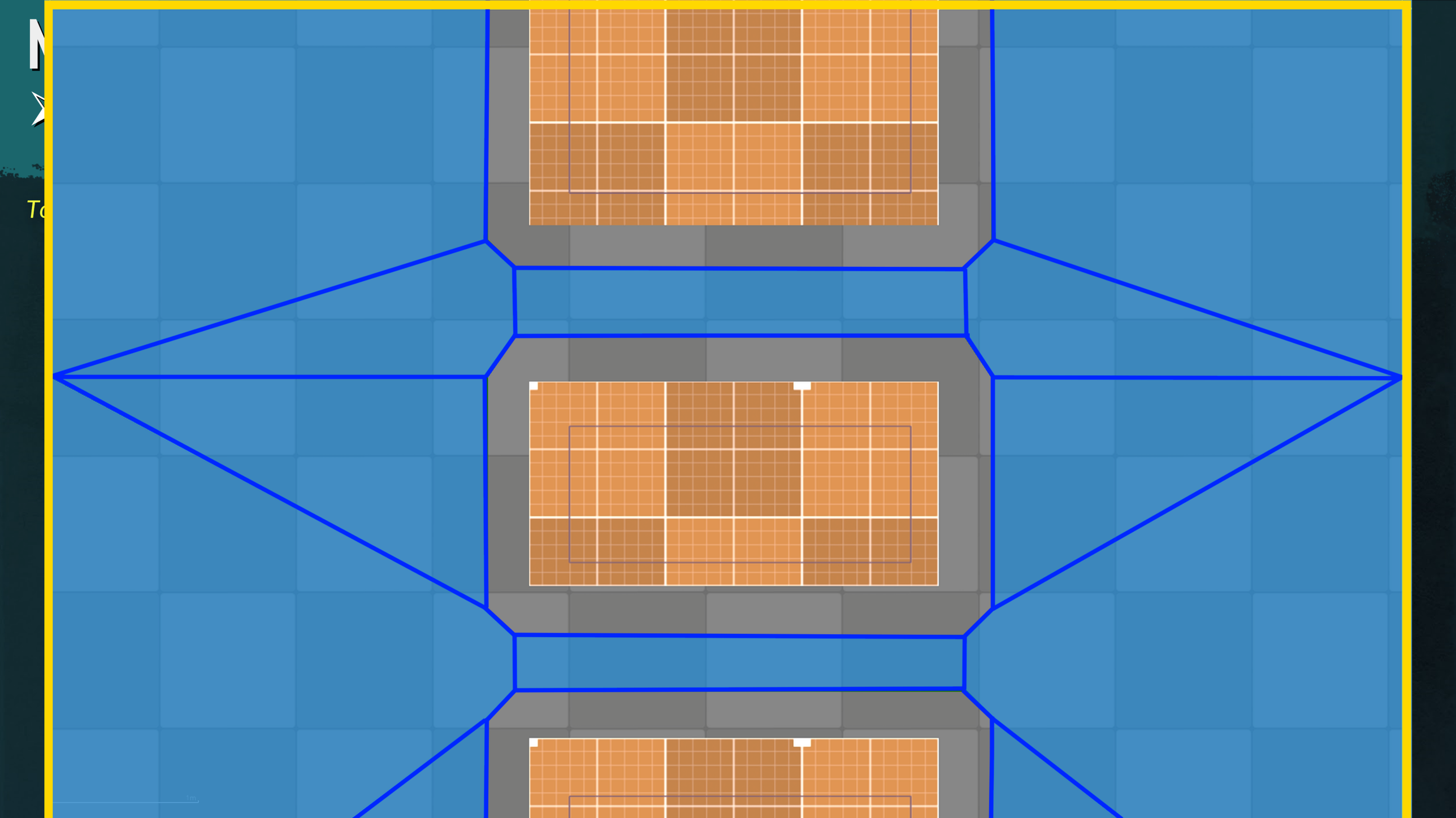
Navigation

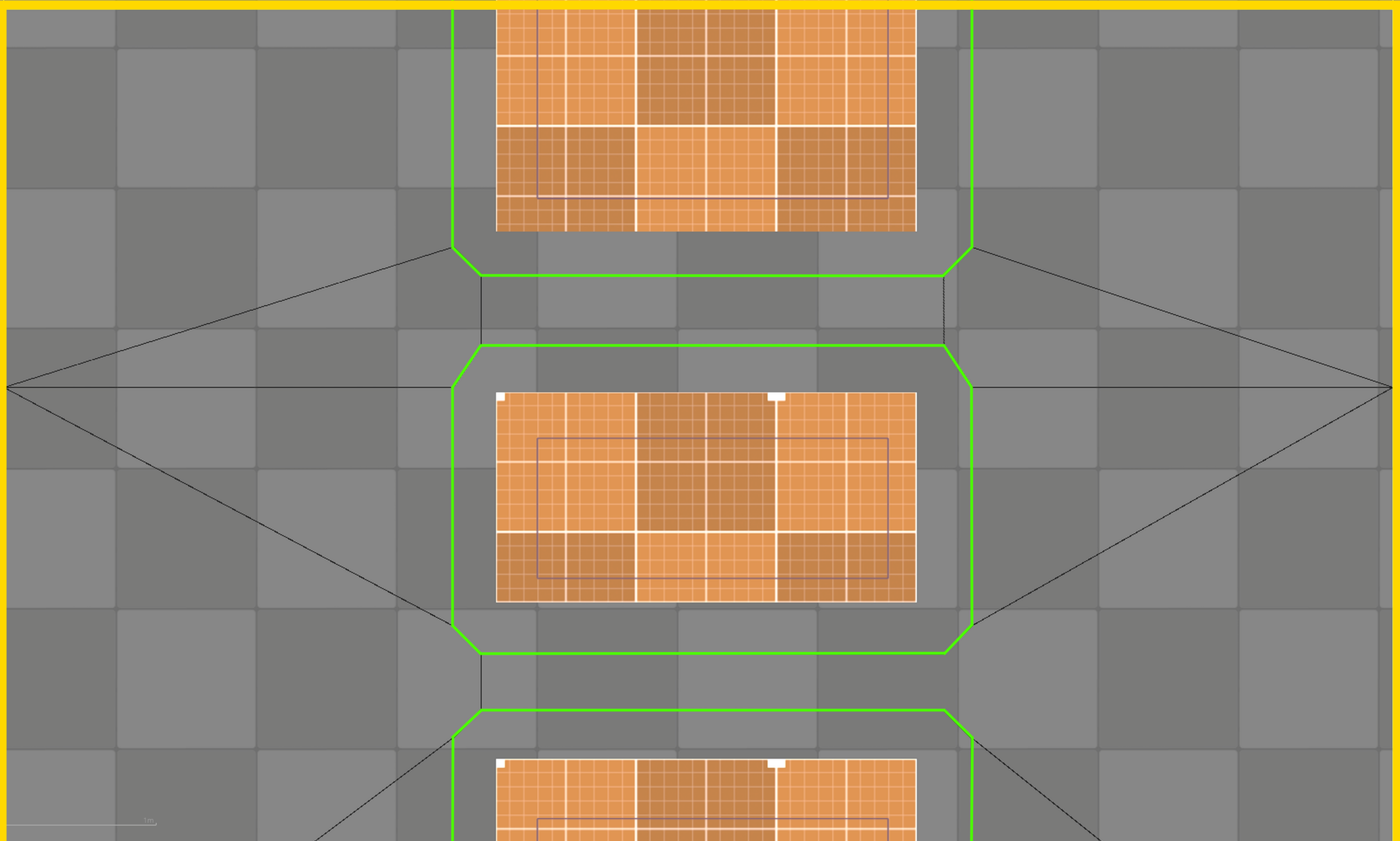
➢ Nav data

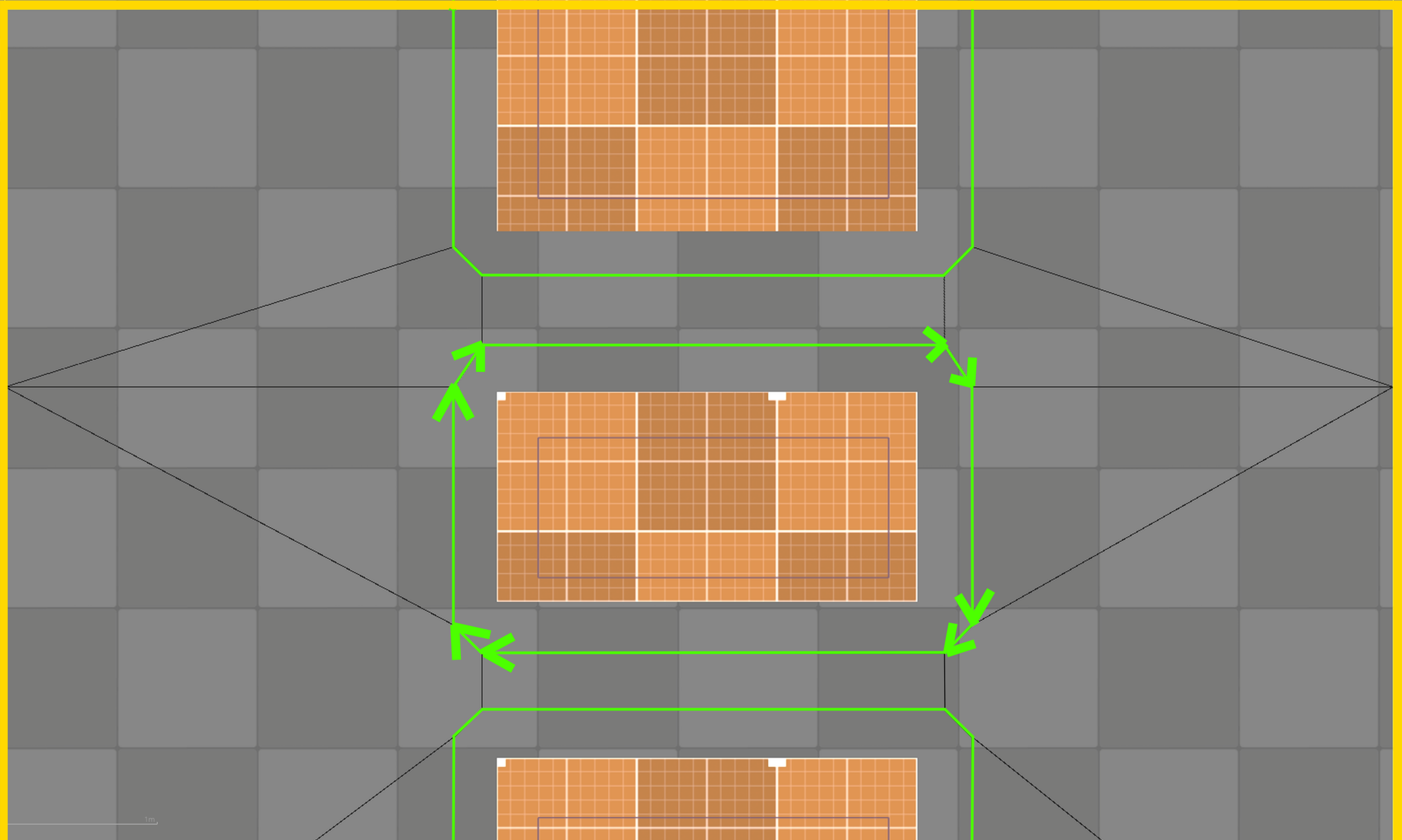
Too many Nav Agents

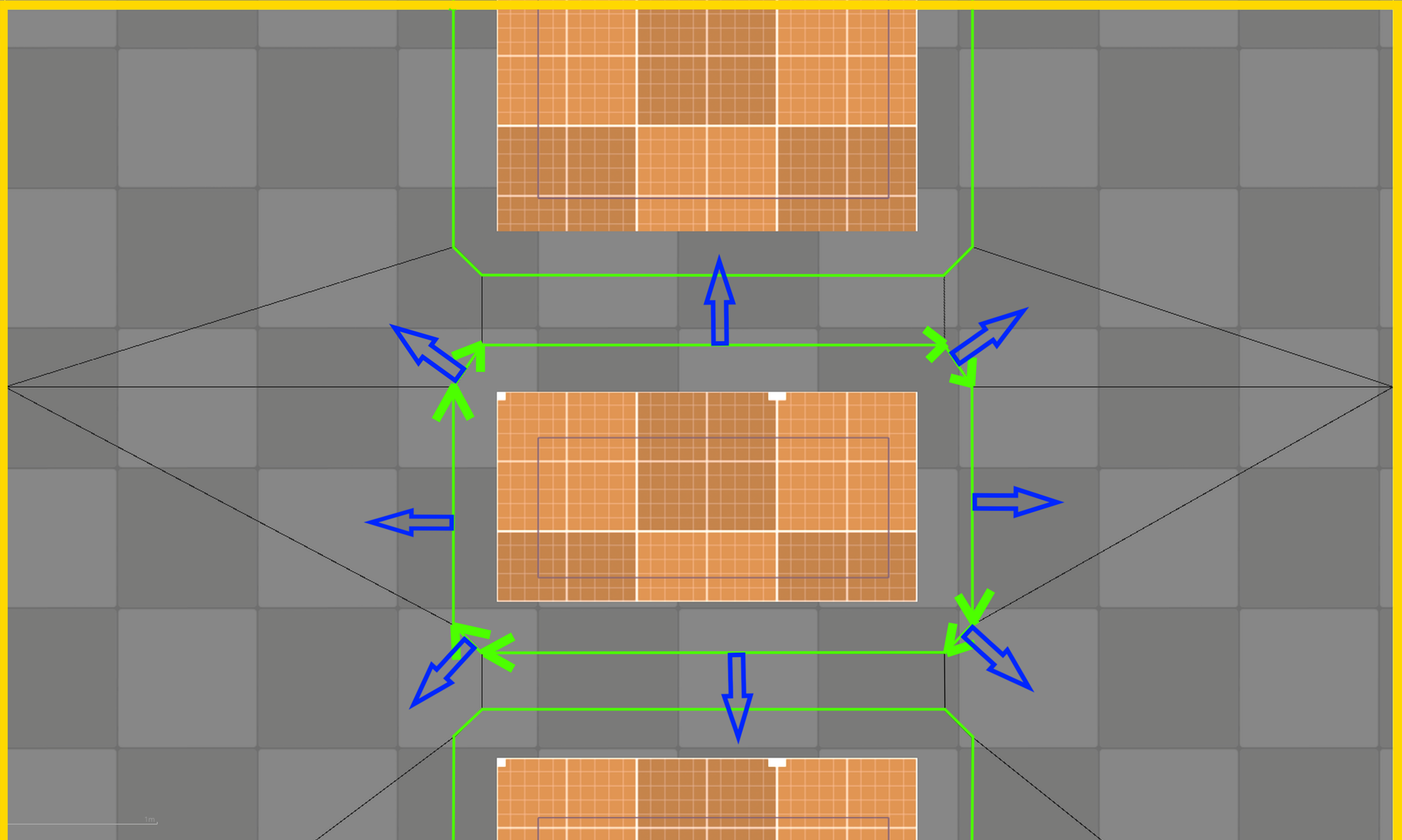




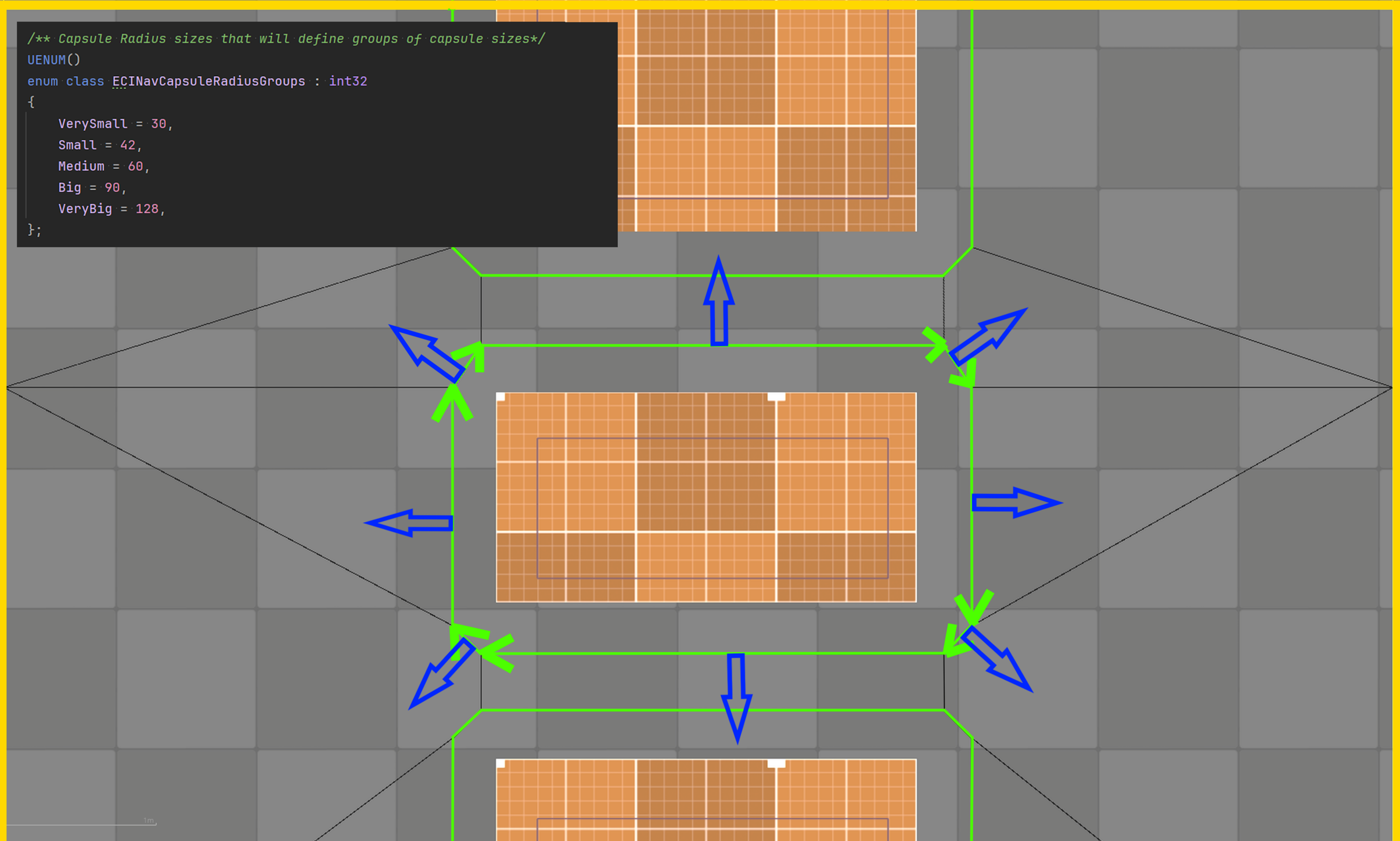


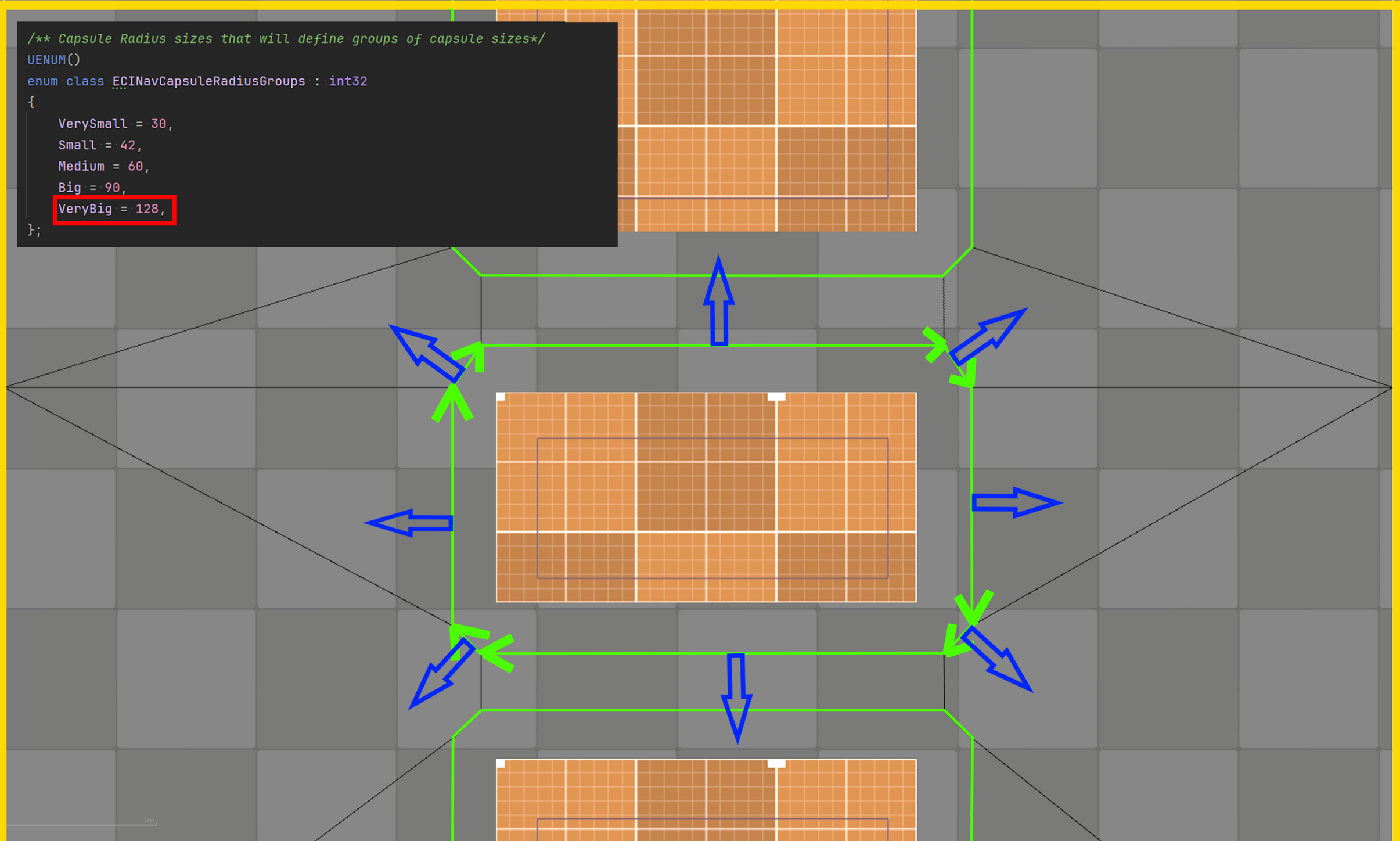


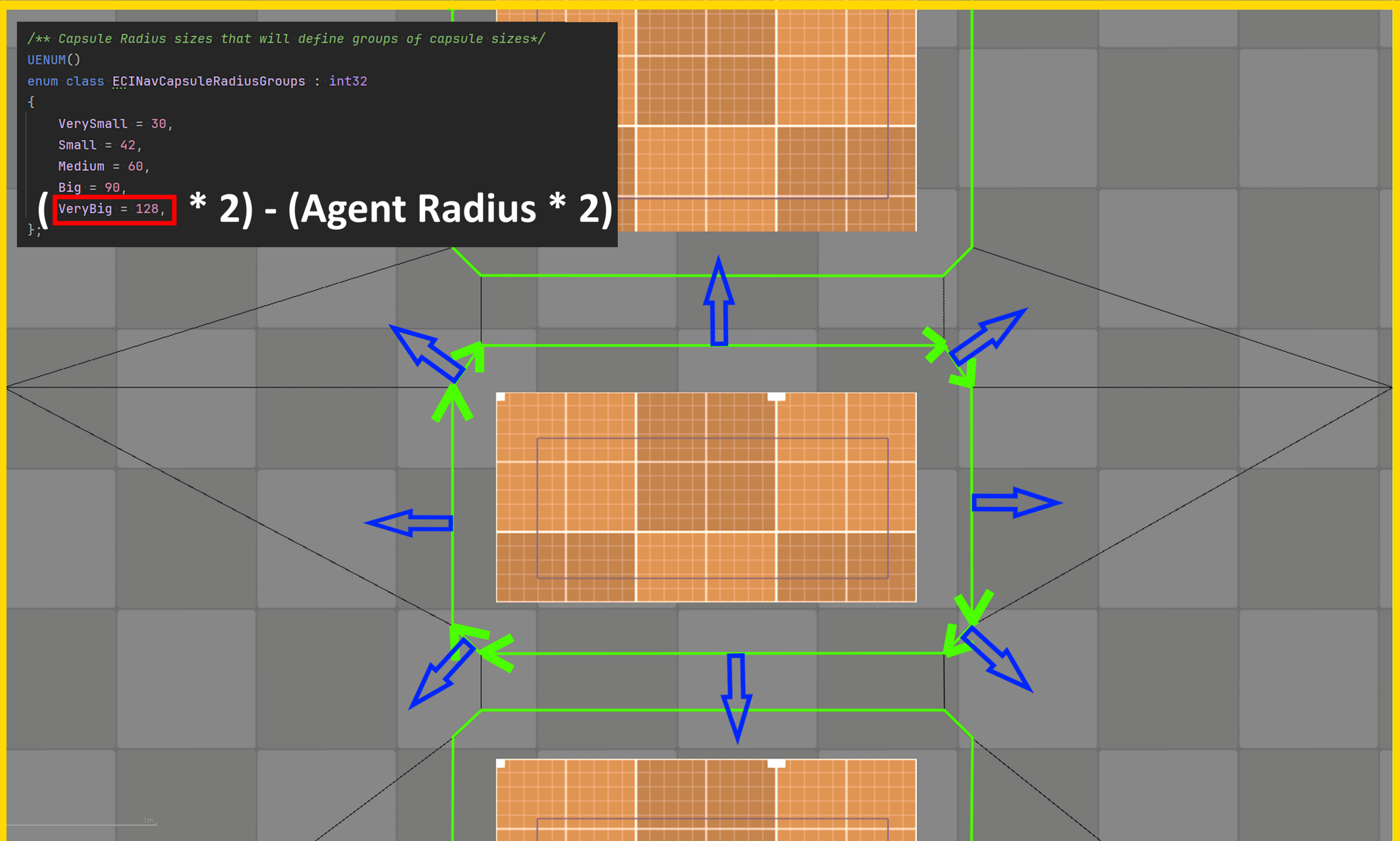


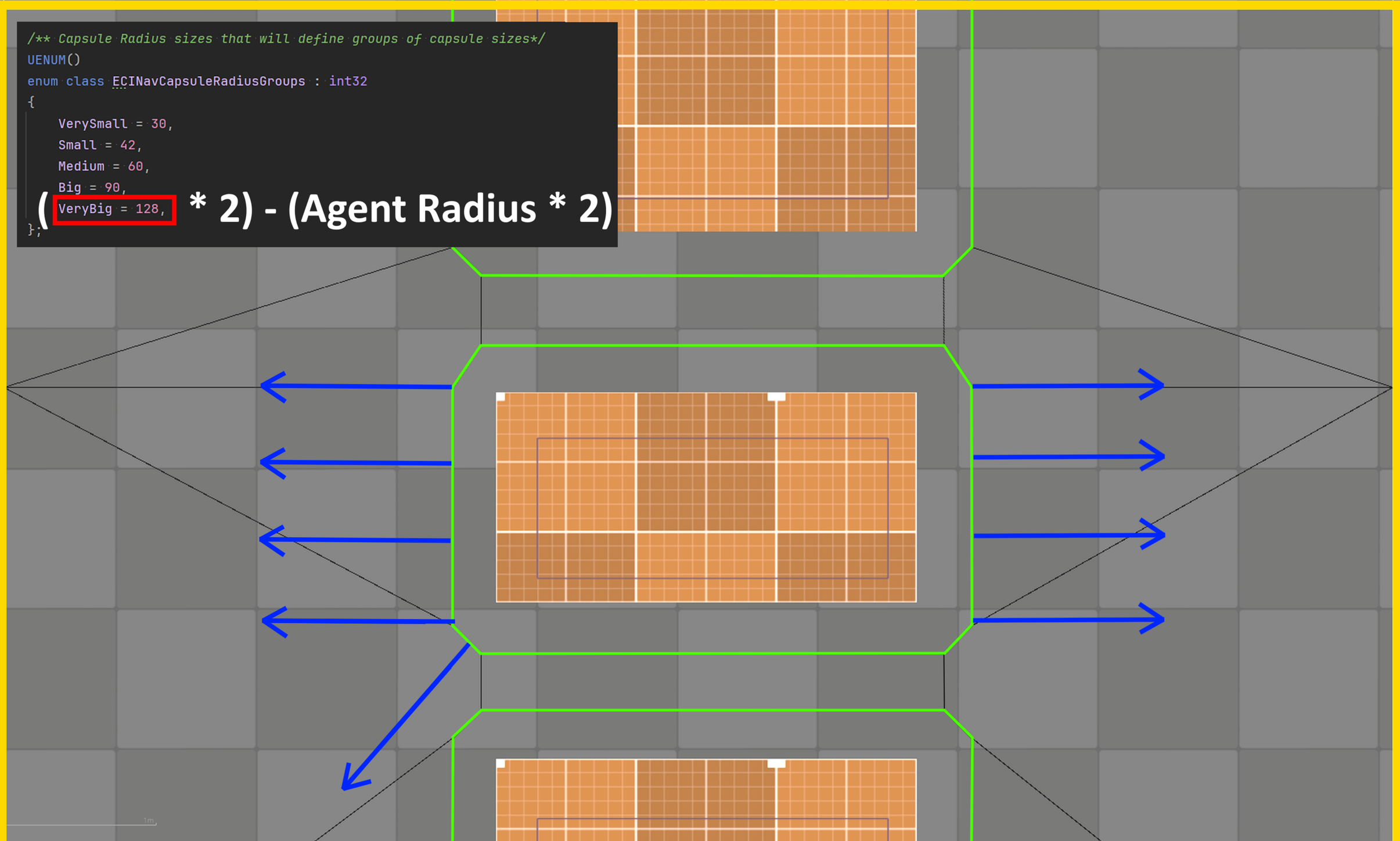


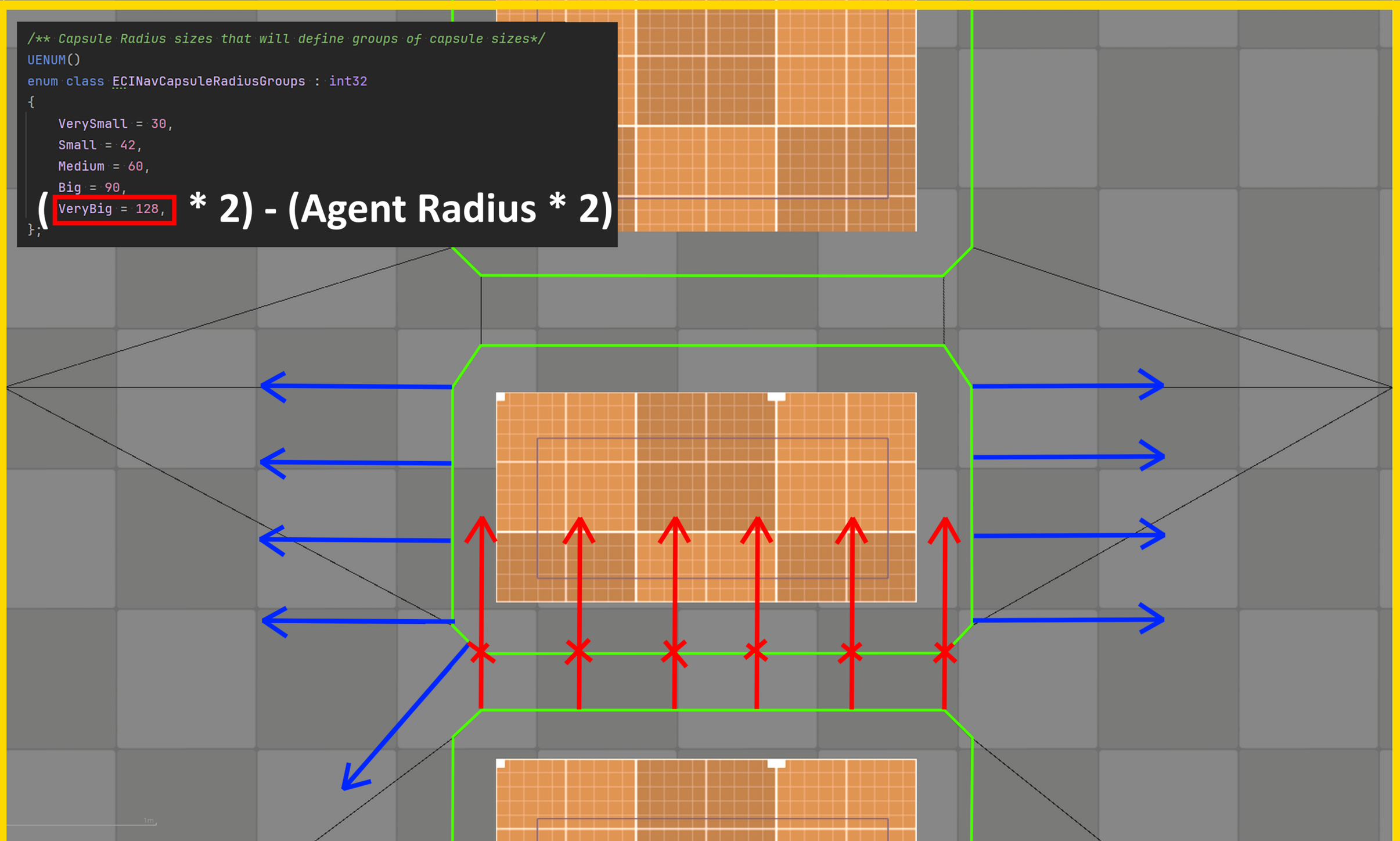
```cpp
/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
    VerySmall = 30,
    Small = 42,
    Medium = 60,
    Big = 90,
    VeryBig = 128,
};
```

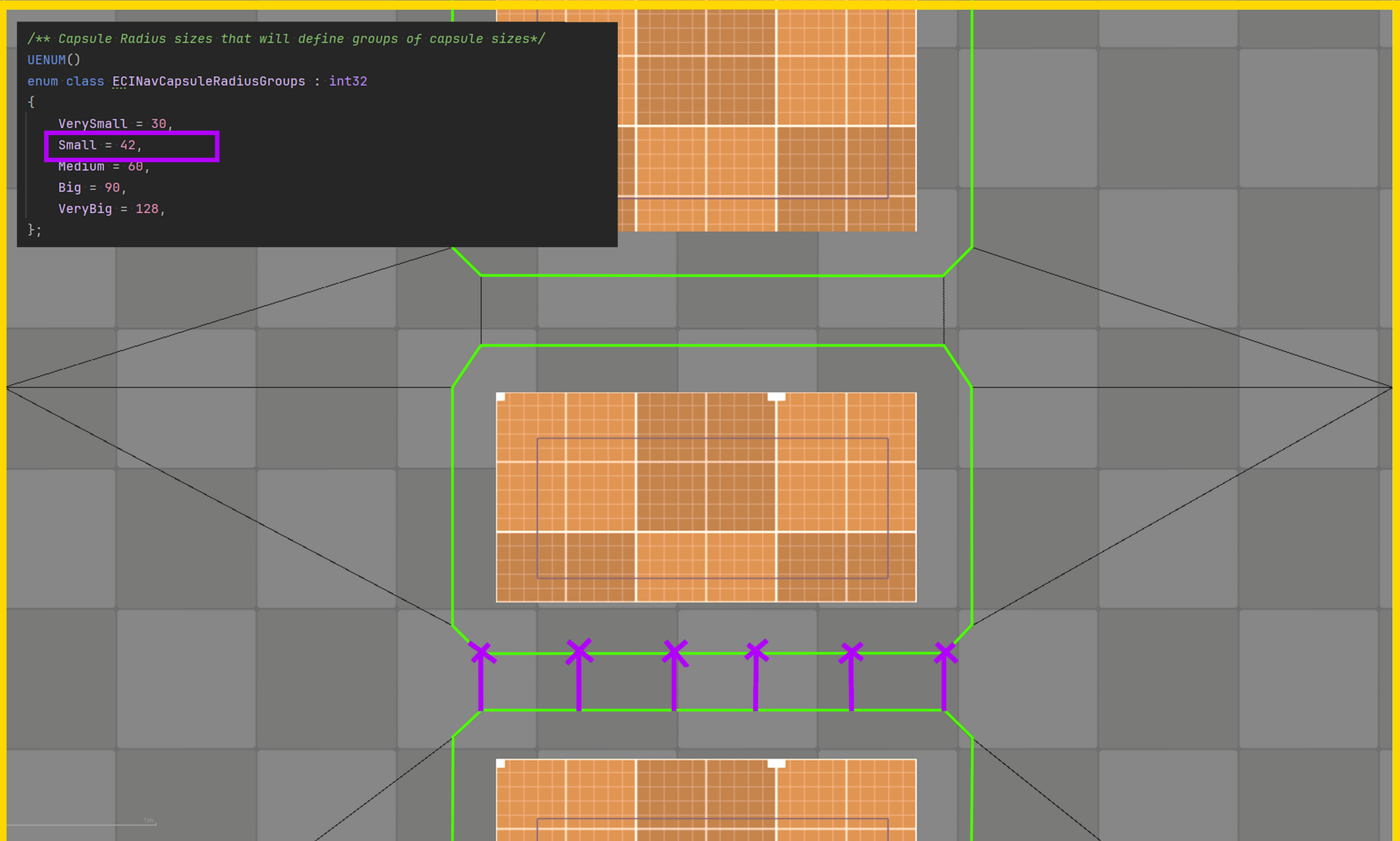



/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};

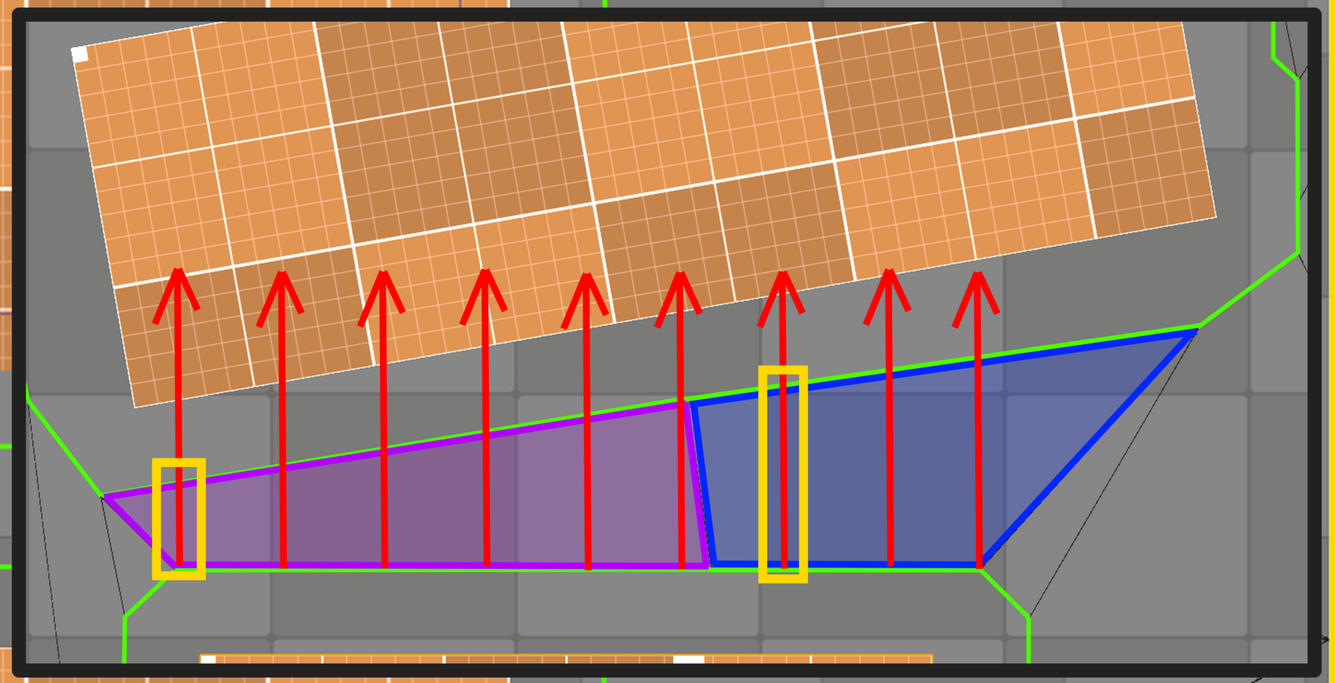

/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
(* 2) - (Agent Radius * 2)


/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
(VeryBig * 2) - (Agent Radius * 2)

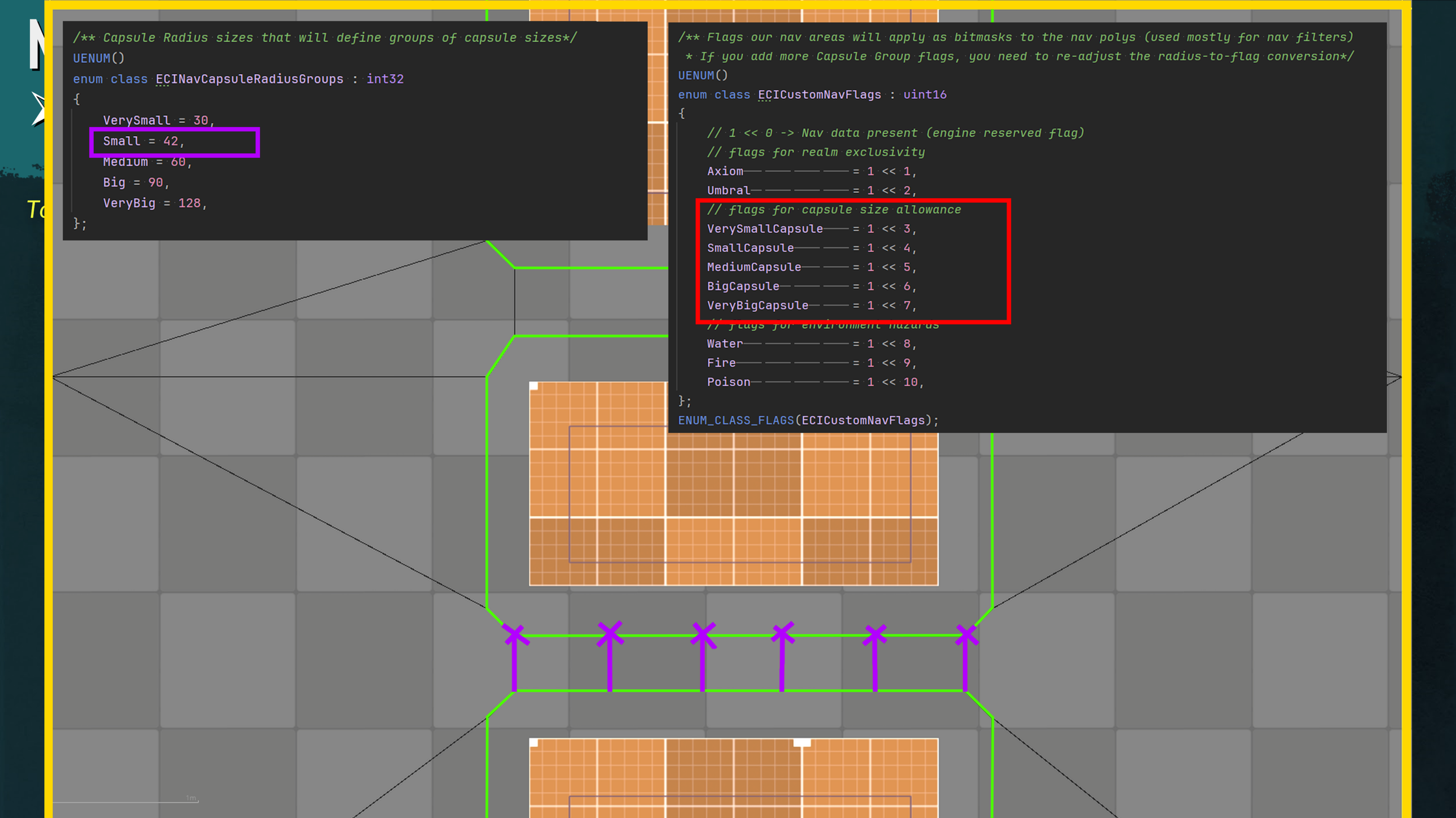

/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
(VeryBig * 2) - (Agent Radius * 2)


/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};

```cpp
/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
    VerySmall = 30,
    Small = 42,
    Medium = 60,
    Big = 90,
    VeryBig = 128,
};
```



/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
/** Flags our nav areas will apply as bitmasks to the nav polys (used mostly for nav filters)
* If you add more Capsule Group flags, you need to re-adjust the radius-to-flag conversion*/
UENUM()
enum class ECICustomNavFlags : uint16
{
// 1 << 0 -> Nav data present (engine reserved flag)
// flags for realm exclusivity
Axiom = 1 << 1,
Umbral = 1 << 2,
// flags for capsule size allowance
VerySmallCapsule = 1 << 3,
SmallCapsule = 1 << 4,
MediumCapsule = 1 << 5,
BigCapsule = 1 << 6,
VeryBigCapsule = 1 << 7,
// flags for environment hazards
Water = 1 << 8,
Fire = 1 << 9,
Poison = 1 << 10,
};
ENUM_CLASS_FLAGS(ECICustomNavFlags);


/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
/** Flags our nav areas will apply as bitmasks to the nav polys (used mostly for nav filters)
* If you add more Capsule Group flags, you need to re-adjust the radius-to-flag conversion*/
UENUM()
enum class ECICustomNavFlags : uint16
{
// 1 << 0 -> Nav data present (engine reserved flag)
// flags for realm exclusivity
Axiom = 1 << 1,
Umbral = 1 << 2,
// flags for capsule size allowance
VerySmallCapsule = 1 << 3,
SmallCapsule = 1 << 4,
MediumCapsule = 1 << 5,
BigCapsule = 1 << 6,
VeryBigCapsule = 1 << 7,
Water = 1 << 8,
Fire = 1 << 9,
Poison = 1 << 10,
};
ENUM_CLASS_FLAGS(ECICustomNavFlags);

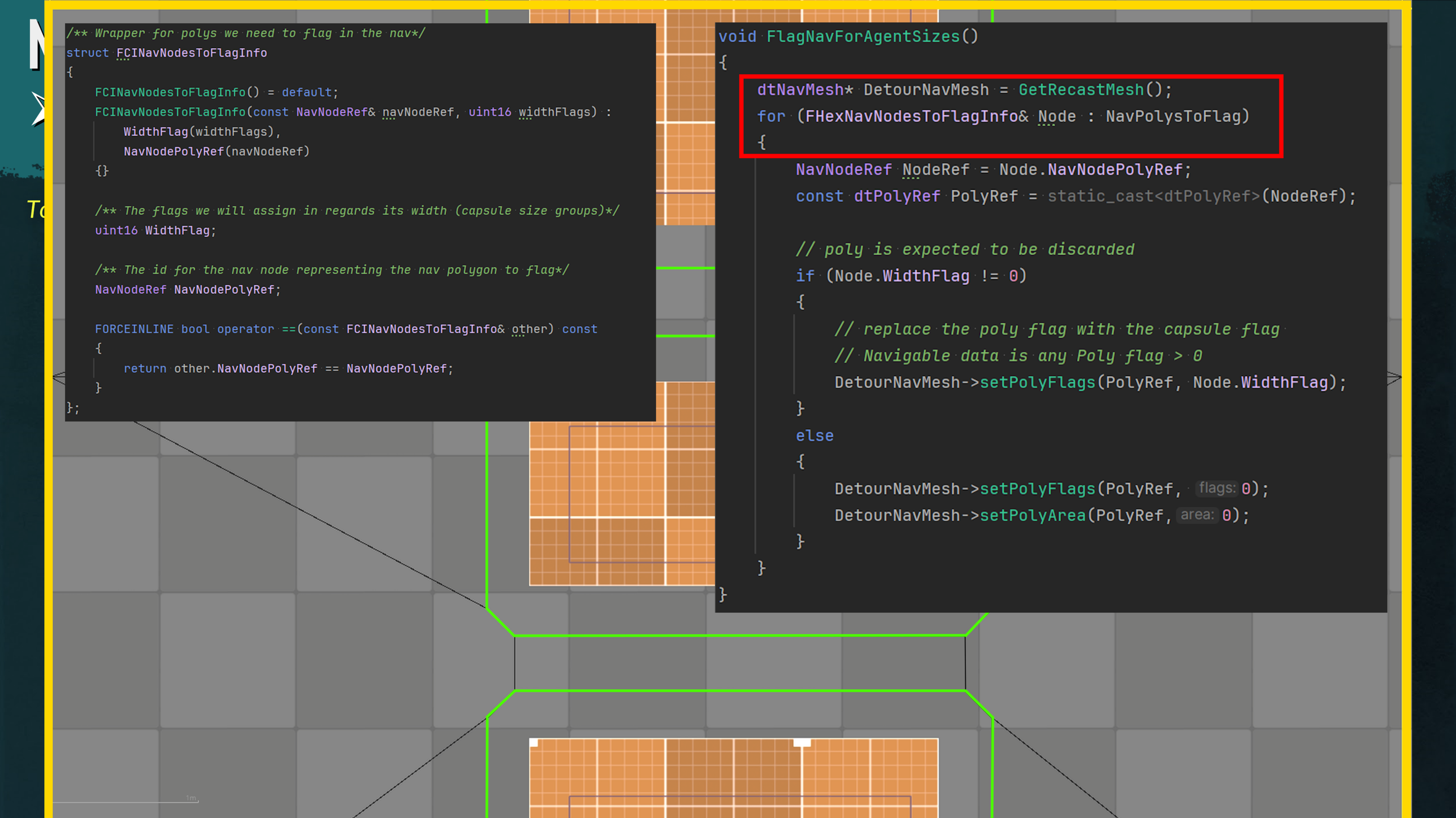
/** Capsule Radius sizes that will define groups of capsule sizes*/
UENUM()
enum class ECINavCapsuleRadiusGroups : int32
{
VerySmall = 30,
Small = 42,
Medium = 60,
Big = 90,
VeryBig = 128,
};
/** Flags our nav areas will apply as bitmasks to the nav polys (used mostly for nav filters)
* If you add more Capsule Group flags, you need to re-adjust the radius-to-flag conversion*/
UENUM()
enum class ECICustomNavFlags : uint16
{
// 1 << 0 -> Nav data present (engine reserved flag)
// flags for realm exclusivity
Axiom = 1 << 1,
Umbral = 1 << 2,
// flags for capsule size allowance
VerySmallCapsule = 1 << 3,
SmallCapsule = 1 << 4,
MediumCapsule = 1 << 5,
BigCapsule = 1 << 6,
VeryBigCapsule = 1 << 7,
// flags for environment hazards
Water = 1 << 8,
Fire = 1 << 9,
Poison = 1 << 10,
};
ENUM_CLASS_FLAGS(ECICustomNavFlags);

/** Flags our nav areas will apply as bitmasks to the nav polys (used mostly for nav filters)
* If you add more Capsule Group flags, you need to re-adjust the radius-to-flag conversion*/
UENUM()
enum class ECICustomNavFlags : uint16
{
// 1 << 0 -> Nav data present (engine reserved flag)
// flags for realm exclusivity
Axiom = 1 << 1,
Umbral = 1 << 2,
// flags for capsule size allowance
VerySmallCapsule = 1 << 3,
SmallCapsule = 1 << 4,
MediumCapsule = 1 << 5,
BigCapsule = 1 << 6,
VeryBigCapsule = 1 << 7,
// flags for environment hazards
Water = 1 << 8,
Fire = 1 << 9,
Poison = 1 << 10,
};
ENUM_CLASS_FLAGS(ECICustomNavFlags);
NAV NODE POLY REF

```cpp
/** Wrapper for polys we need to flag in the nav*/
struct FCINavNodesToFlagInfo
{
    FCINavNodesToFlagInfo() = default;
    FCINavNodesToFlagInfo(const NavNodeRef& navNodeRef, uint16 widthFlags) :
        WidthFlag(widthFlags),
        NavNodePolyRef(navNodeRef)
    {}

    /** The flags we will assign in regards its width (capsule size groups)*/
    uint16 WidthFlag;

    /** The id for the nav node representing the nav polygon to flag*/
    NavNodeRef NavNodePolyRef;

    FORCEINLINE bool operator ==(const FCINavNodesToFlagInfo& other) const
    {
        return other.NavNodePolyRef == NavNodePolyRef;
    }
};
```

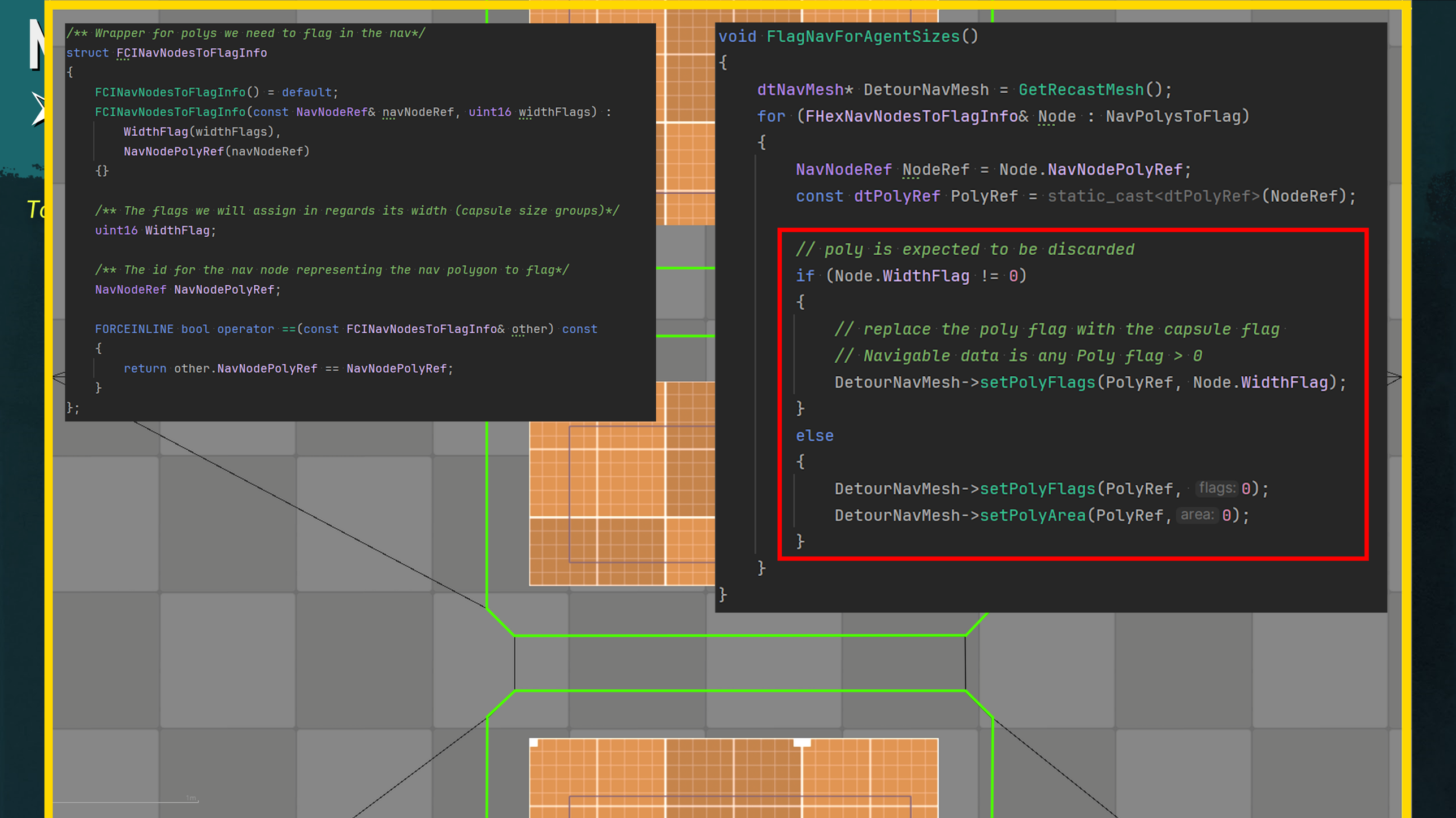
```cpp
/** Flags our nav areas will apply as bitmasks to the nav polys (used mostly for nav filters)
 * If you add more Capsule Group flags, you need to re-adjust the radius-to-flag conversion*/
UENUM()
enum class ECICustomNavFlags : uint16
{
    // 1 << 0 -> Nav data present (engine reserved flag)
    // flags for realm exclusivity
    Axiom = 1 << 1,
    Umbral = 1 << 2,
    // flags for capsule size allowance
    VerySmallCapsule = 1 << 3,
    SmallCapsule = 1 << 4,
    MediumCapsule = 1 << 5,
    BigCapsule = 1 << 6,
    VeryBigCapsule = 1 << 7,
    // flags for environment hazards
    Water = 1 << 8,
    Fire = 1 << 9,
    Poison = 1 << 10,
};
ENUM_CLASS_FLAGS(ECICustomNavFlags);
```

NAV NODE POLY REF



```cpp
/** Wrapper for polys we need to flag in the nav*/
struct FCINavNodesToFlagInfo
{
    FCINavNodesToFlagInfo() = default;
    FCINavNodesToFlagInfo(const NavNodeRef& navNodeRef, uint16 widthFlags) :
        WidthFlag(widthFlags),
        NavNodePolyRef(navNodeRef)
    {}

    /** The flags we will assign in regards its width (capsule size groups)*/
    uint16 WidthFlag;

    /** The id for the nav node representing the nav polygon to flag*/
    NavNodeRef NavNodePolyRef;

    FORCEINLINE bool operator ==(const FCINavNodesToFlagInfo& other) const
    {
        return other.NavNodePolyRef == NavNodePolyRef;
    }
};
```

```cpp
void FlagNavForAgentSizes()
{
    dtNavMesh* DetourNavMesh = GetRecastMesh();
    for (FHexNavNodesToFlagInfo& Node : NavPolysToFlag)
    {
        NavNodeRef NodeRef = Node.NavNodePolyRef;
        const dtPolyRef PolyRef = static_cast<dtPolyRef>(NodeRef);

        // poly is expected to be discarded
        if (Node.WidthFlag != 0)
        {
            // replace the poly flag with the capsule flag
            // Navigable data is any Poly flag > 0
            DetourNavMesh->setPolyFlags(PolyRef, Node.WidthFlag);
        }
        else
        {
            DetourNavMesh->setPolyFlags(PolyRef, flags: 0);
            DetourNavMesh->setPolyArea(PolyRef, area: 0);
        }
    }
}
```



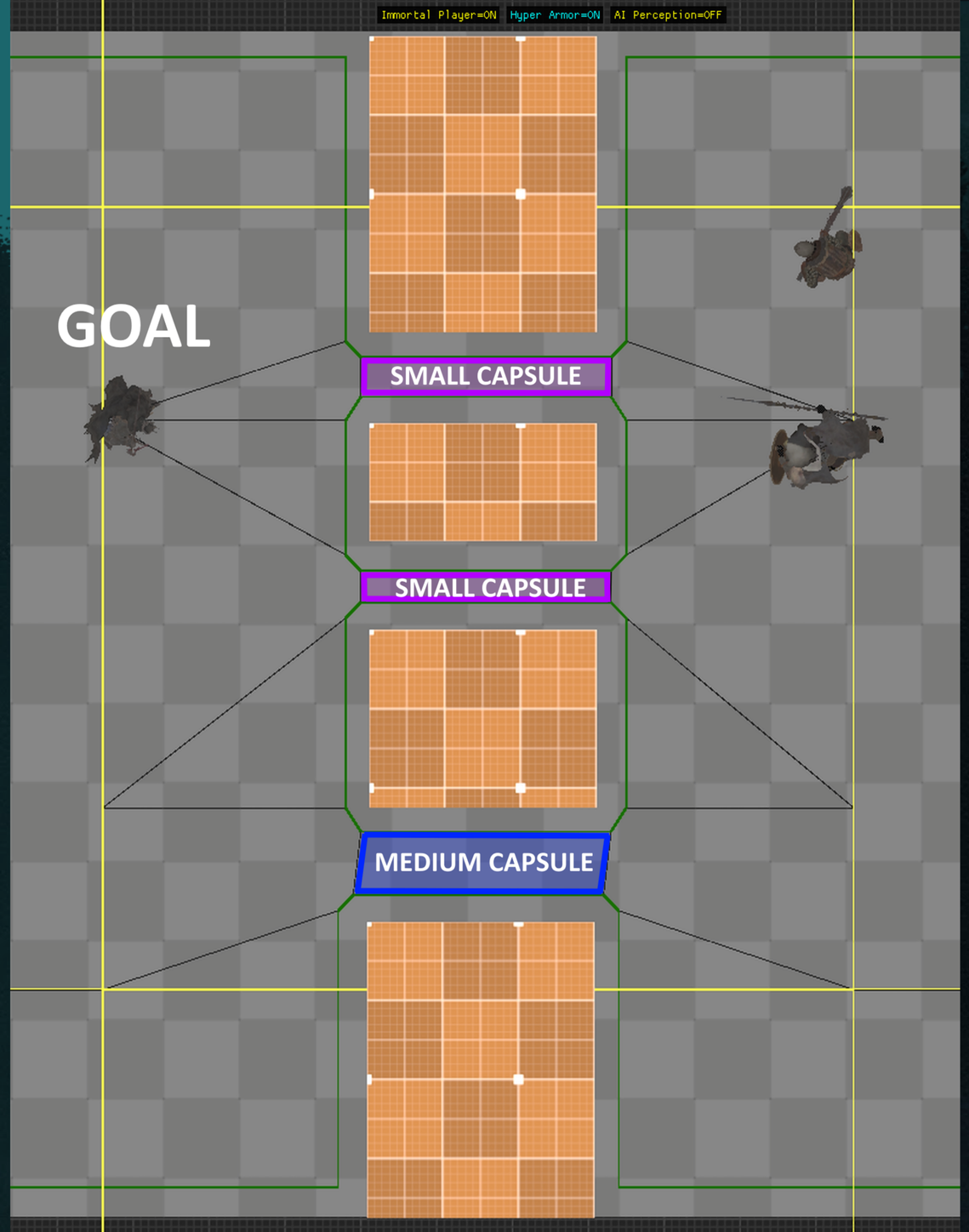
```cpp
/** Wrapper for polys we need to flag in the nav*/
struct FCINavNodesToFlagInfo
{
    FCINavNodesToFlagInfo() = default;
    FCINavNodesToFlagInfo(const NavNodeRef& navNodeRef, uint16 widthFlags) :
        WidthFlag(widthFlags),
        NavNodePolyRef(navNodeRef)
    {}

    /** The flags we will assign in regards its width (capsule size groups)*/
    uint16 WidthFlag;

    /** The id for the nav node representing the nav polygon to flag*/
    NavNodeRef NavNodePolyRef;

    FORCEINLINE bool operator ==(const FCINavNodesToFlagInfo& other) const
    {
        return other.NavNodePolyRef == NavNodePolyRef;
    }
};
```

```cpp
void FlagNavForAgentSizes()
{
    dtNavMesh* DetourNavMesh = GetRecastMesh();
    for (FHexNavNodesToFlagInfo& Node : NavPolysToFlag)
    {
        NavNodeRef NodeRef = Node.NavNodePolyRef;
        const dtPolyRef PolyRef = static_cast<dtPolyRef>(NodeRef);

        // poly is expected to be discarded
        if (Node.WidthFlag != 0)
        {
            // replace the poly flag with the capsule flag
            // Navigable data is any Poly flag > 0
            DetourNavMesh->setPolyFlags(PolyRef, Node.WidthFlag);
        }
        else
        {
            DetourNavMesh->setPolyFlags(PolyRef, flags: 0);
            DetourNavMesh->setPolyArea(PolyRef, area: 0);
        }
    }
}
```

Navigation

➢ Nav Data

Too many Nav Agents



Navigation

➢ Nav data

Reading corridor data

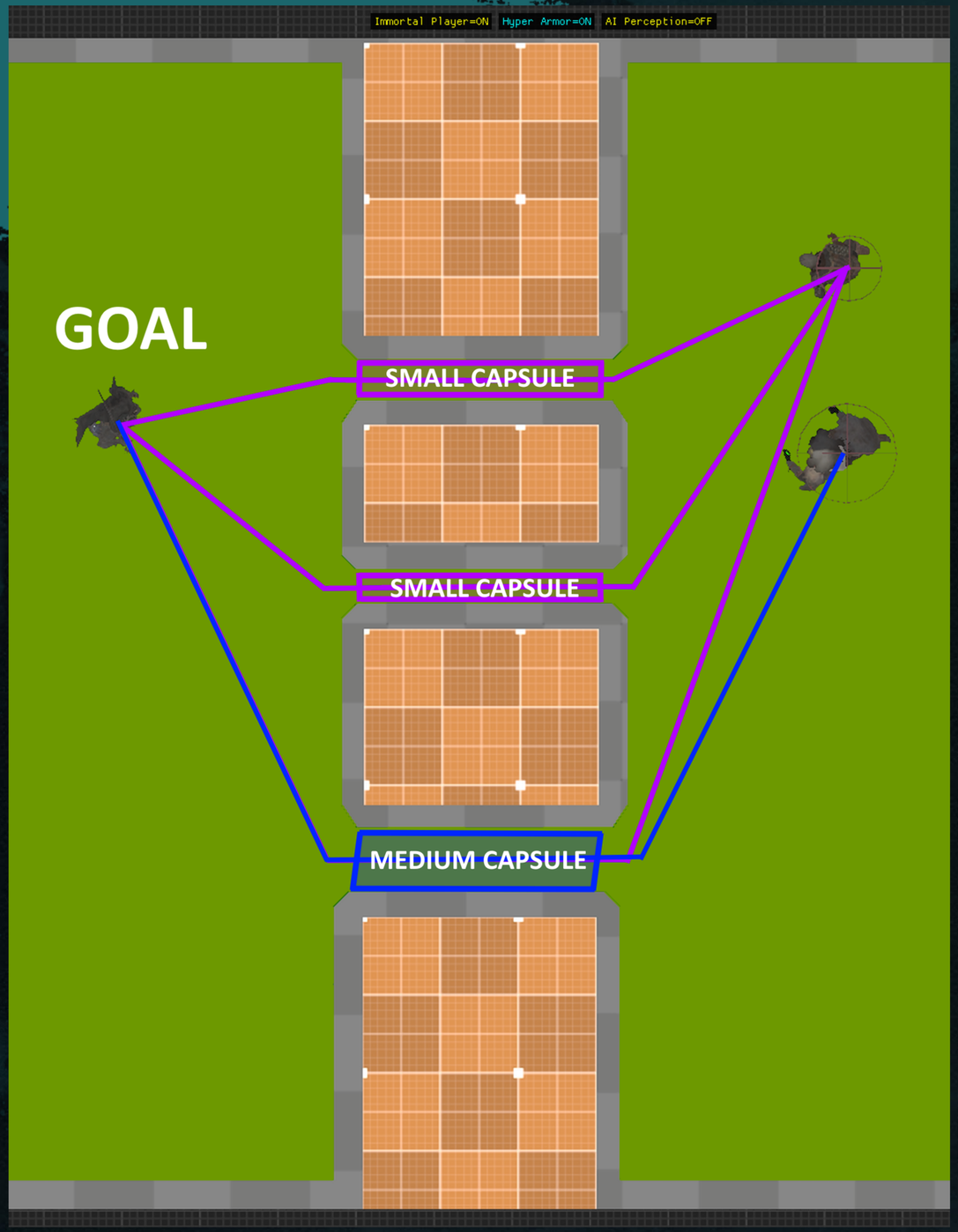
```cpp
void UCINavFilter_AIControllerDefault::InitializeFilter(const ANavigationData& NavData, const UObject* Querier,
    FNavigationQueryFilter& Filter) const
{
    Super::InitializeFilter(NavData, Querier, [&]Filter);
    UpdateForQuerier(Querier, [&]Filter);
}

void UCINavFilter_AIControllerDefault::UpdateForQuerier(const UObject* Querier, FNavigationQueryFilter& Filter) const
{
    const AAIController* Controller = Cast<AAIController>(Querier);
    const APawn* QuerierPawn = ::IsValid(Controller) ? Controller->GetPawn() : nullptr;
    if (QuerierPawn != nullptr)
    {
        // flags to filter out nav areas that are too thin for this character to go through
        FVector2D CapsuleSize = FVector2D::ZeroVector;
        GetActorCapsuleSize(QuerierPawn, [&]CapsuleSize);
        const uint16 CapsuleExcludeFlags = GetExcludedNavFlagsForCapsuleRadius(CapsuleSize.X);

        /**the filter flag will come with the area flag needed for axiom/umbral. add the capsule flag*/
        uint16 NewExcludeFlags = Filter.GetExcludeFlags() | CapsuleExcludeFlags;

        Filter.SetExcludeFlags(NewExcludeFlags);
    }
}

TSubclassOf<UNavigationQueryFilter> UCINavFilter_AIControllerDefault::GetSimpleFilterForAgent(const UObject& querier) const
{
    return TSubclassOf<UCINavFilter_AIControllerDefault>();
}
```

```cpp
moveReq.SetNavigationFilter(taskData.AIController->GetDefaultNavigationFilterClass());
```

Navigation

➢ Nav data

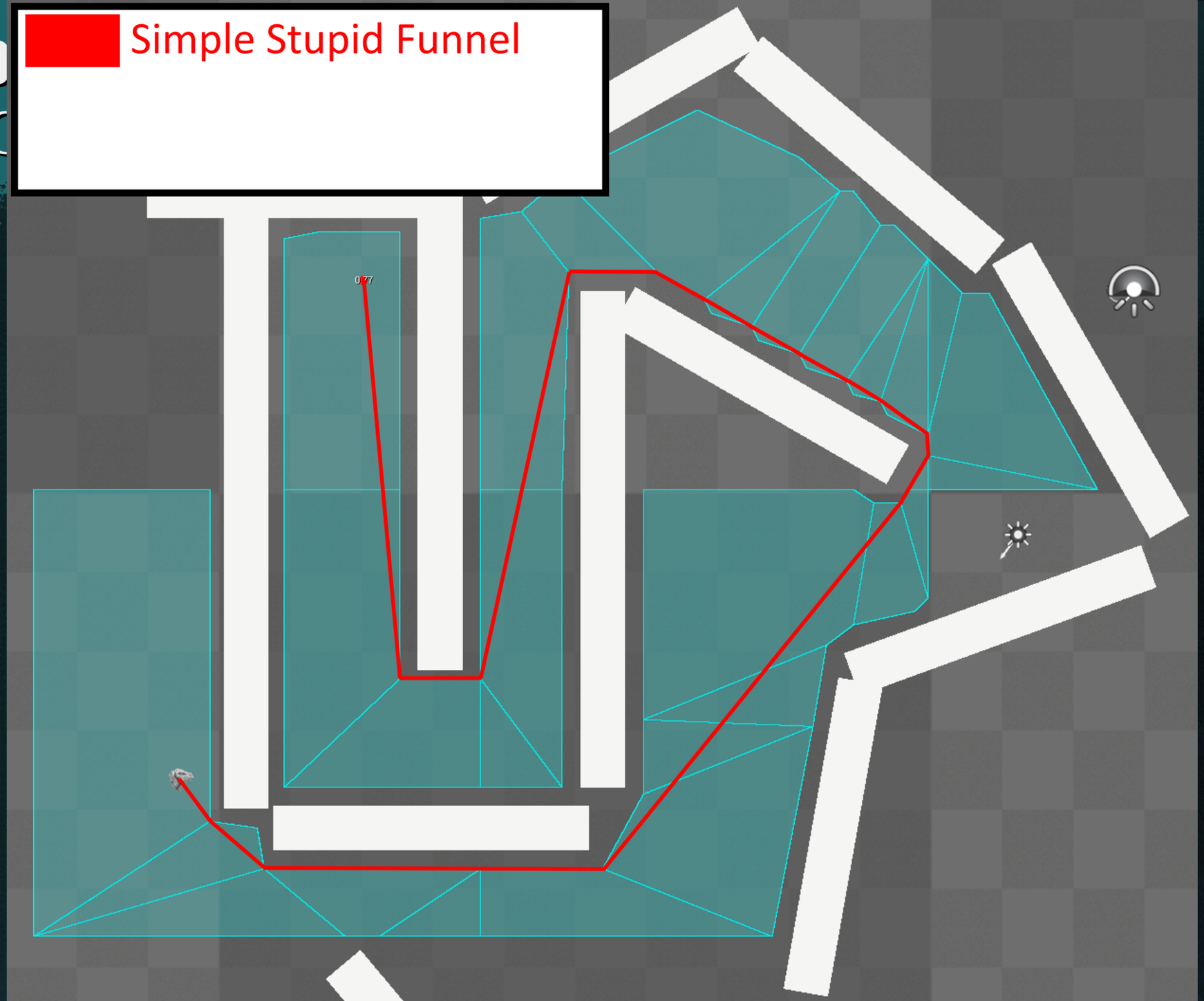
Reading corridor data



Navigatio

➢ Path C

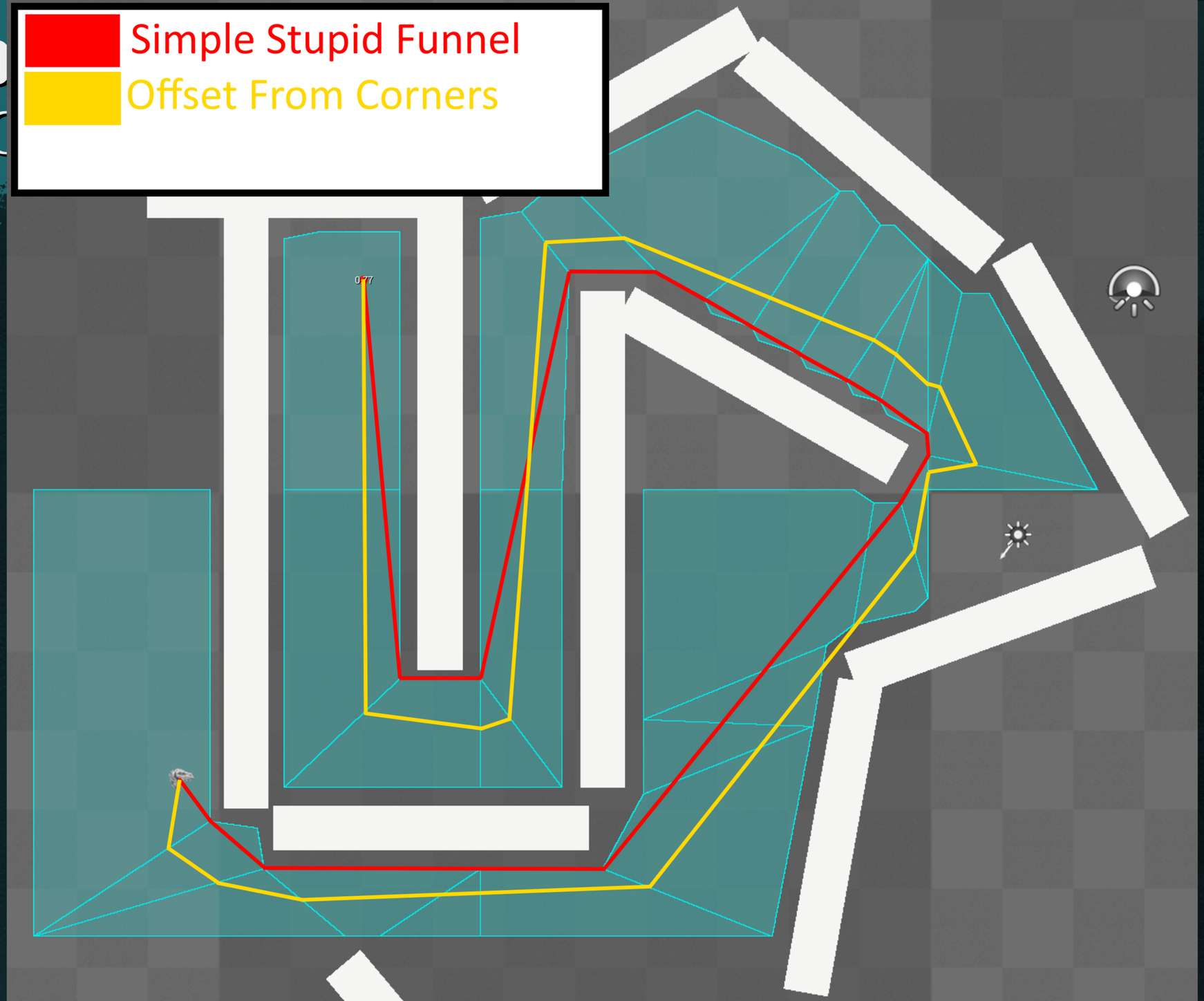
Fit all capsules



Navigatio

➢ Path C

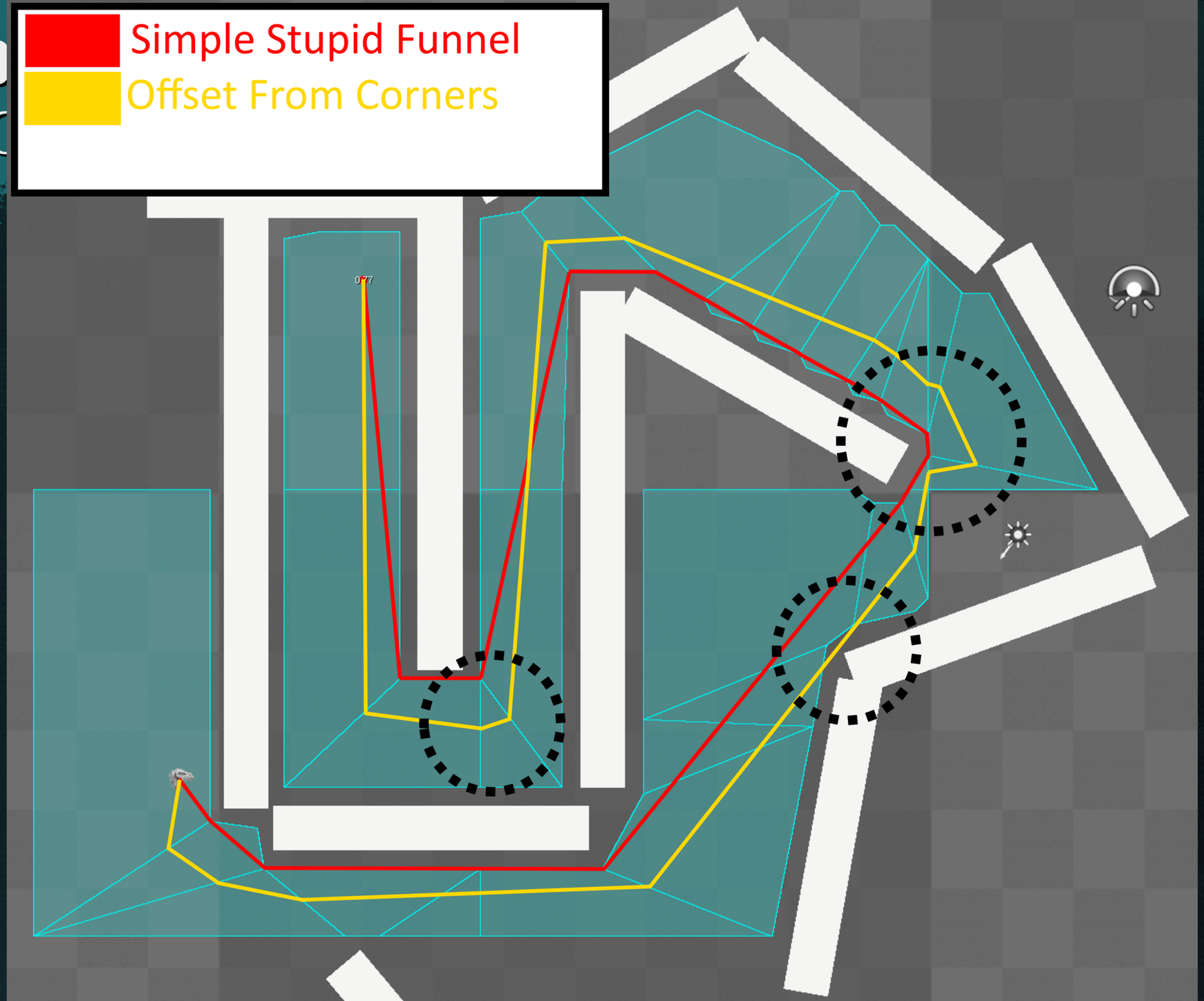
Fit all capsules



Navigatio

➢ Path C

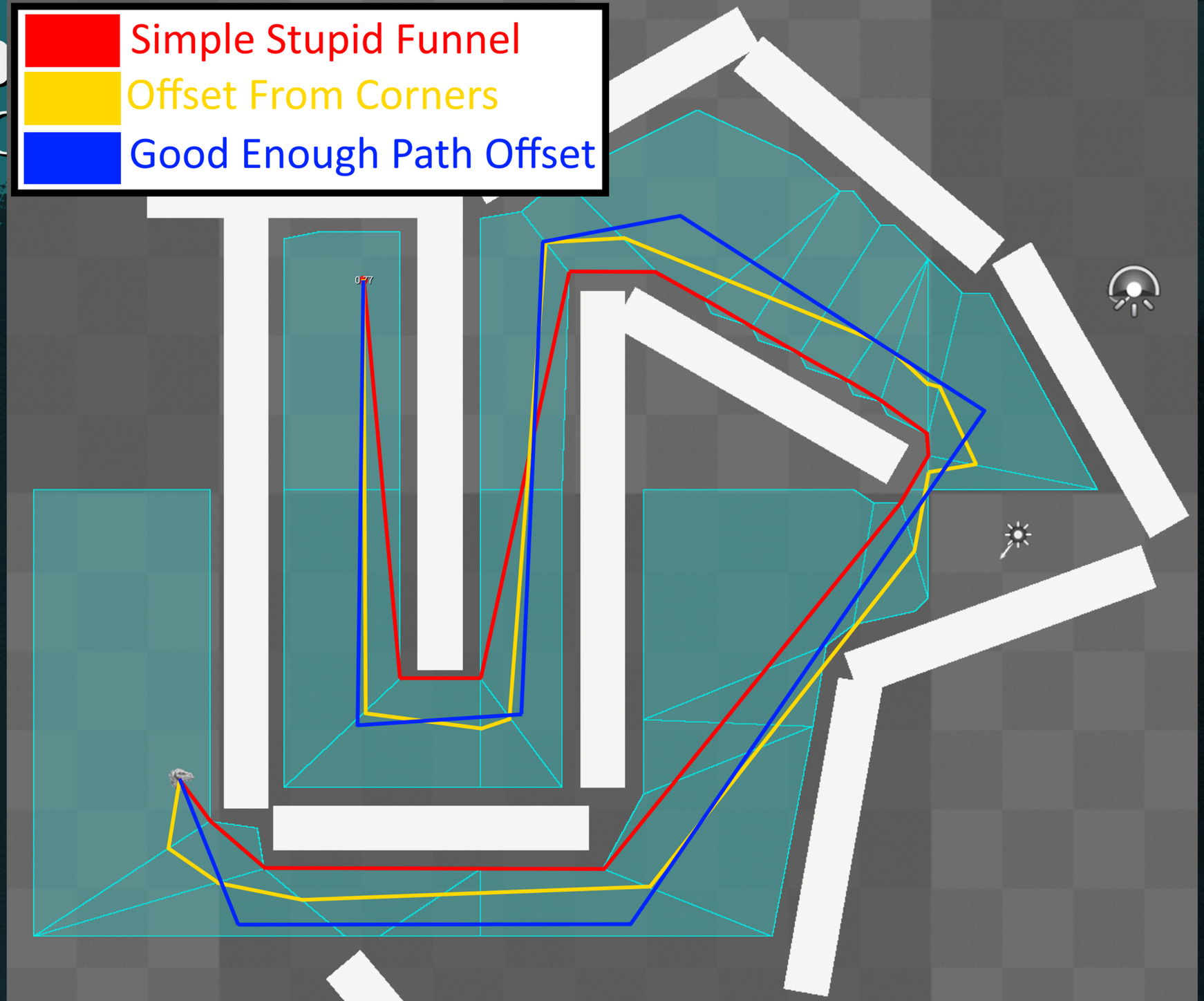
Fit all capsules



Navigatio

➢ Path C

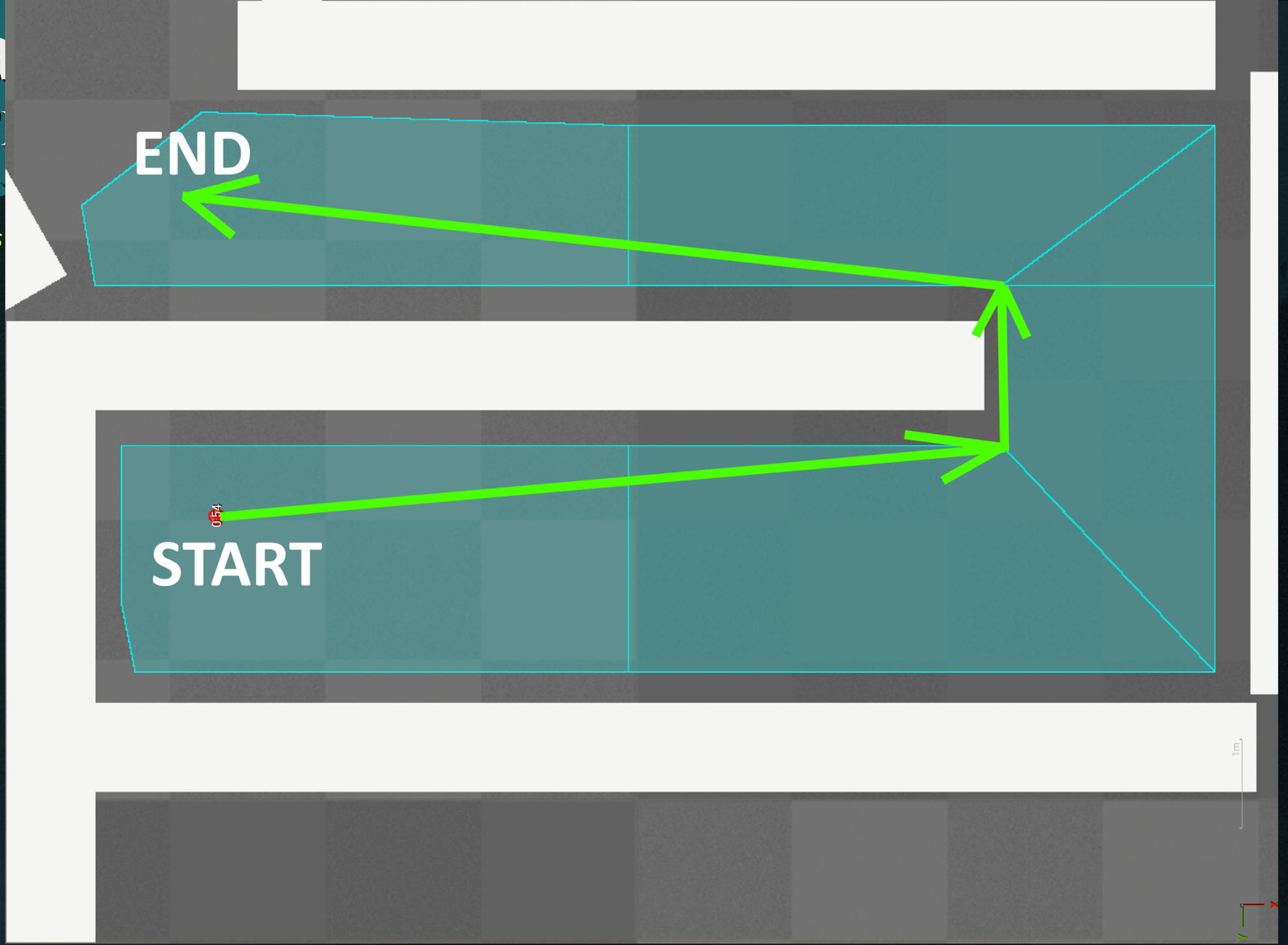
Fit all capsules



Naviga

➢ PAT

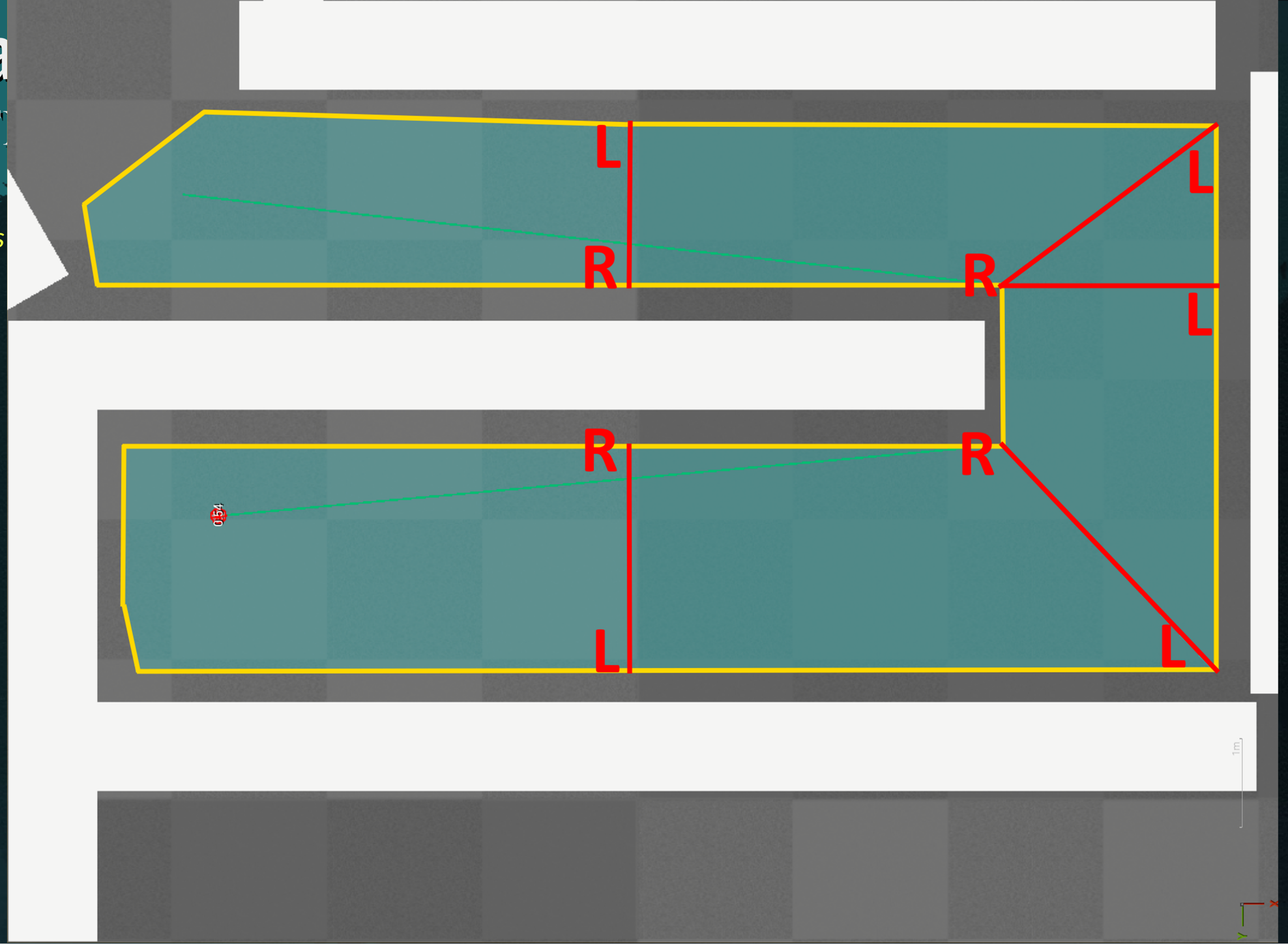
Fit all capsules



Naviga

➢ PAT

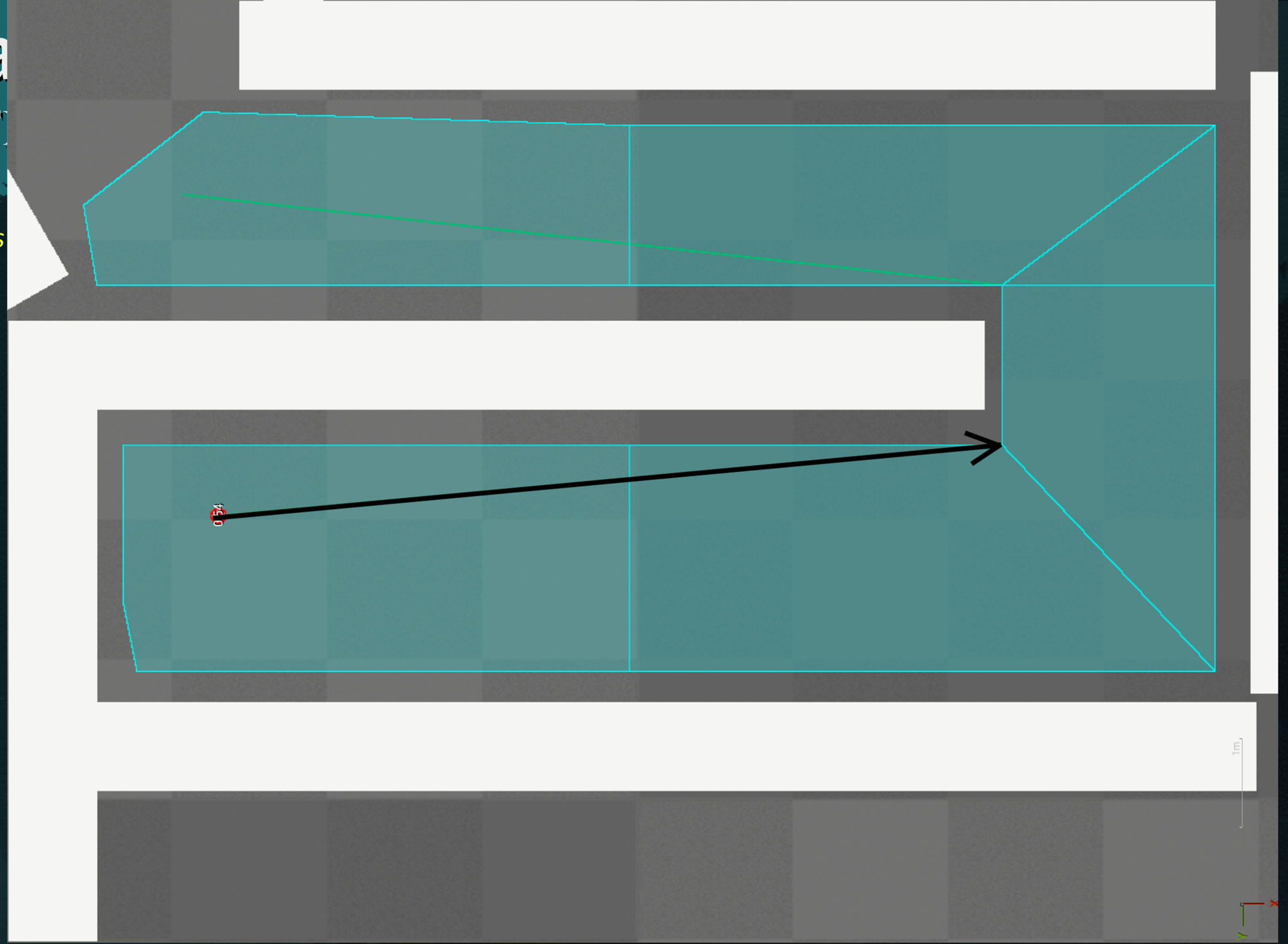
Fit all capsules



Naviga

➢ PAT

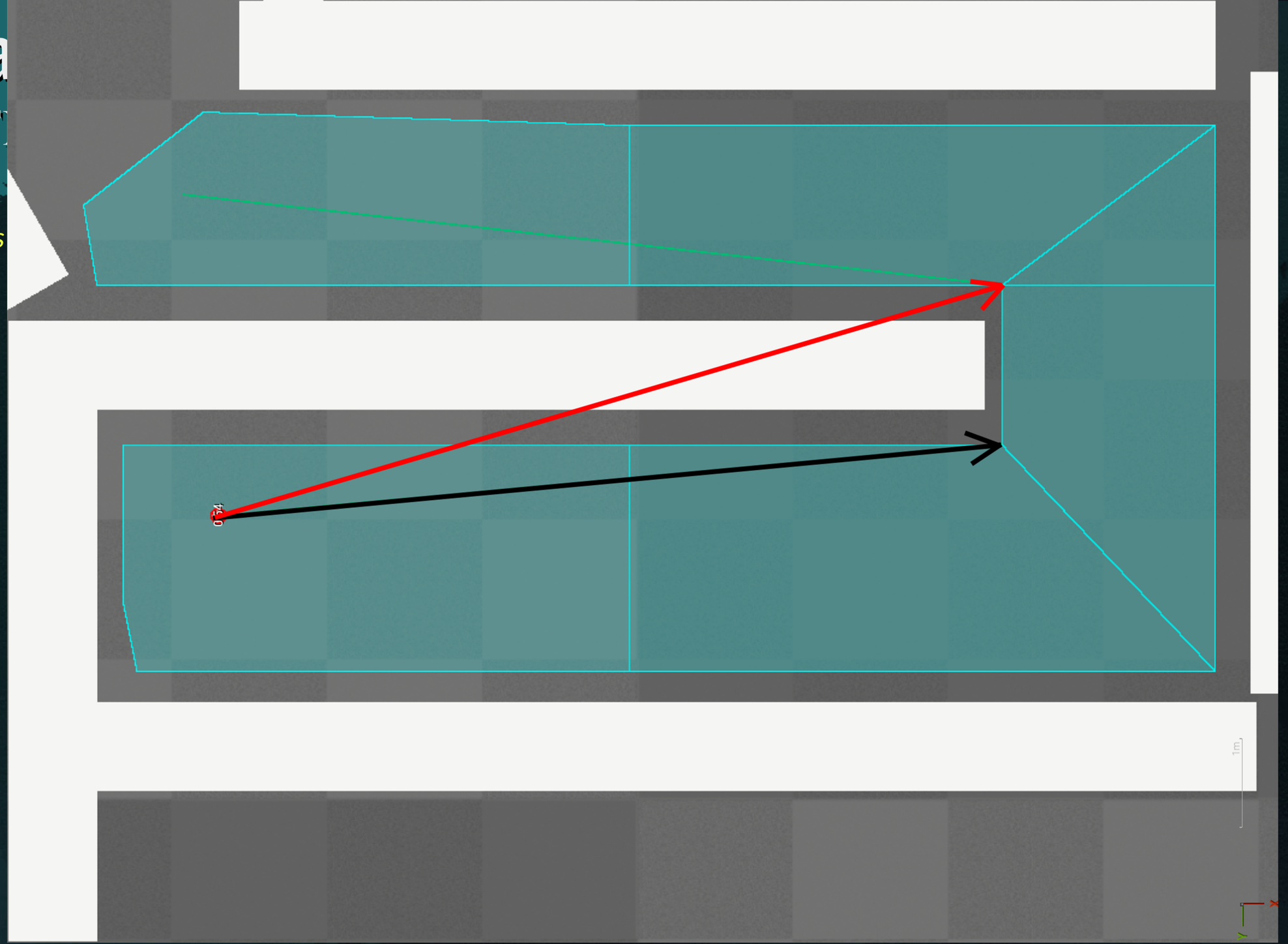
Fit all capsules



Naviga

➢ PAT

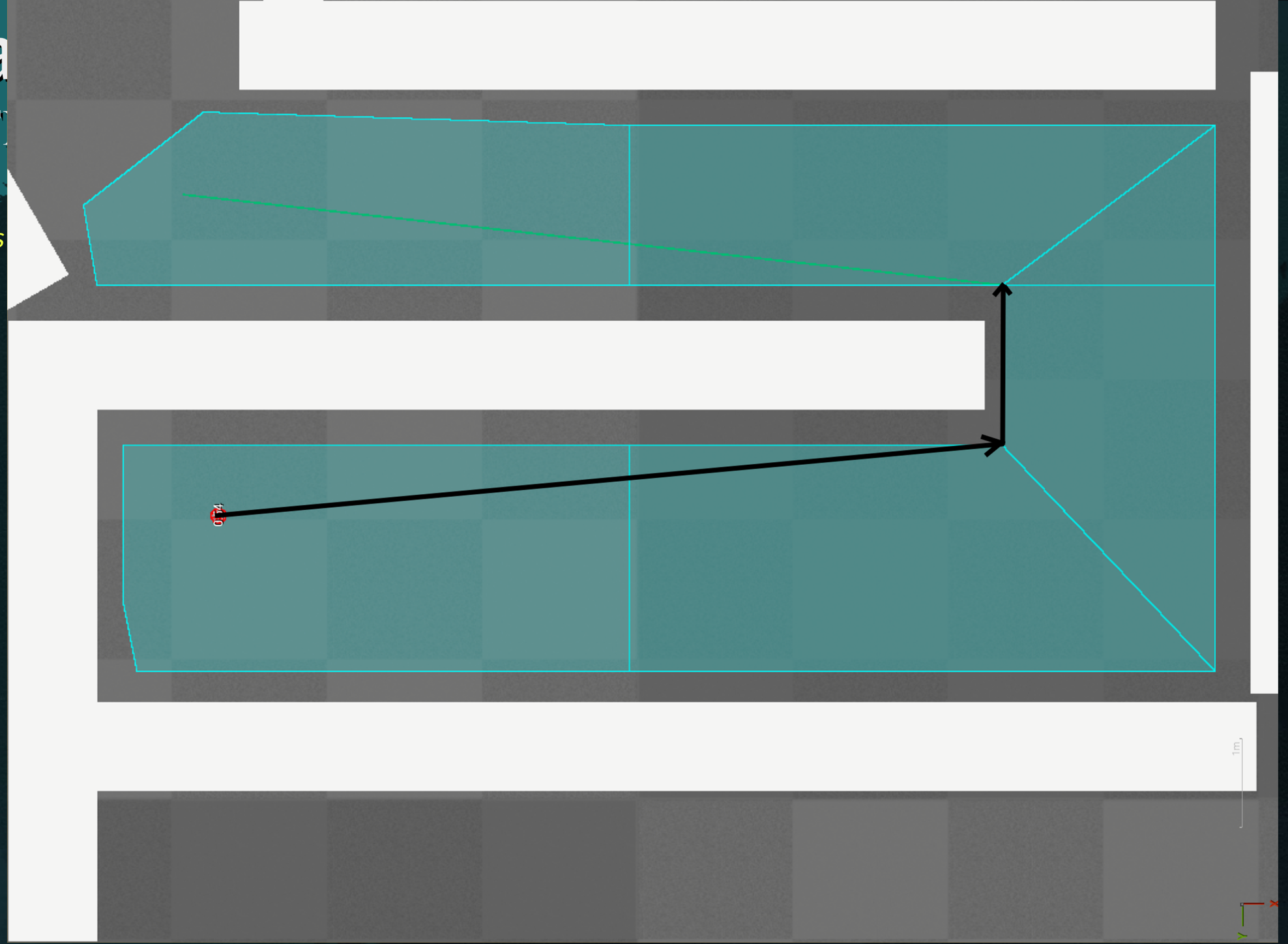
Fit all capsules



Naviga

➢ PAT

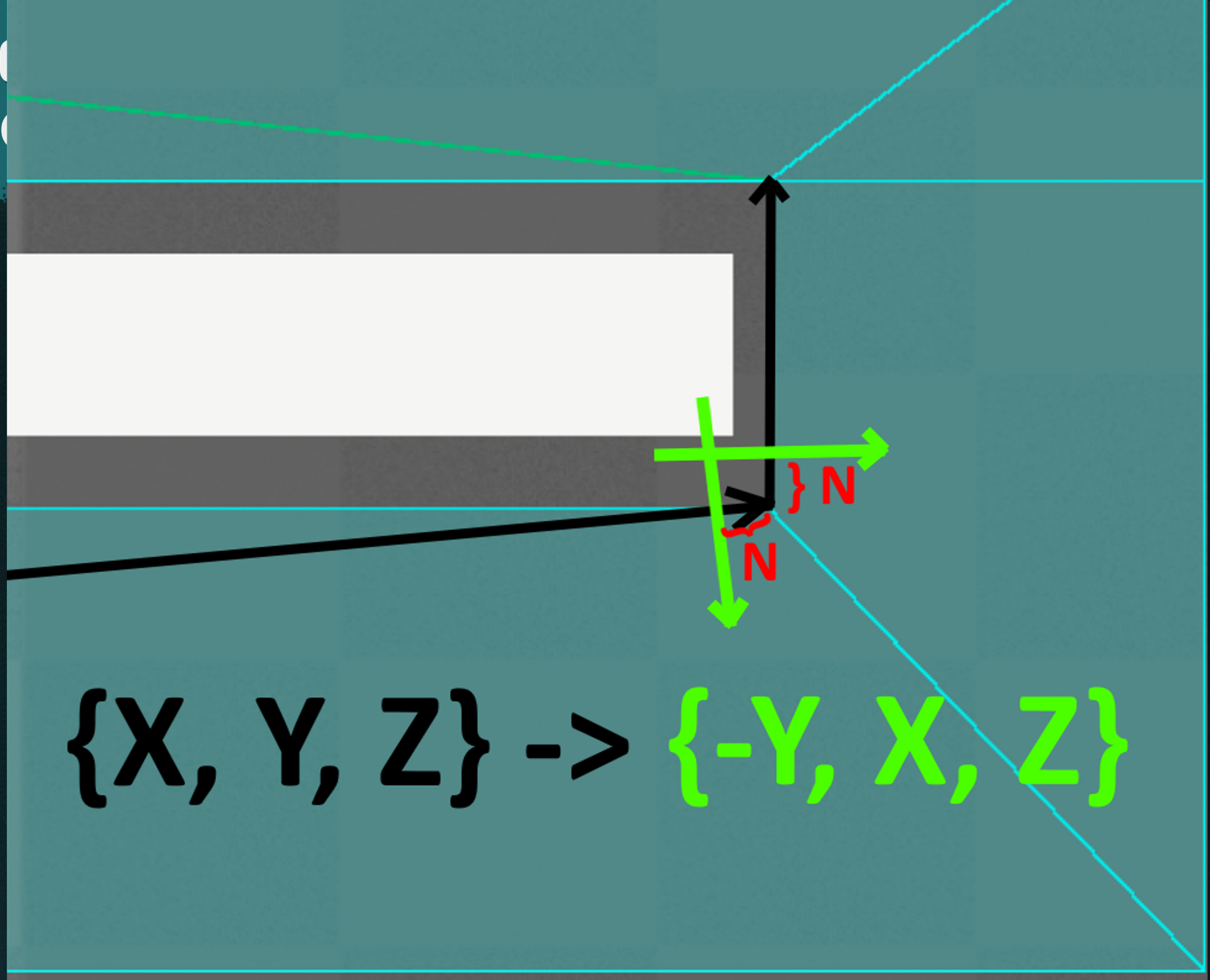
Fit all capsules



Navigati

➢ Path

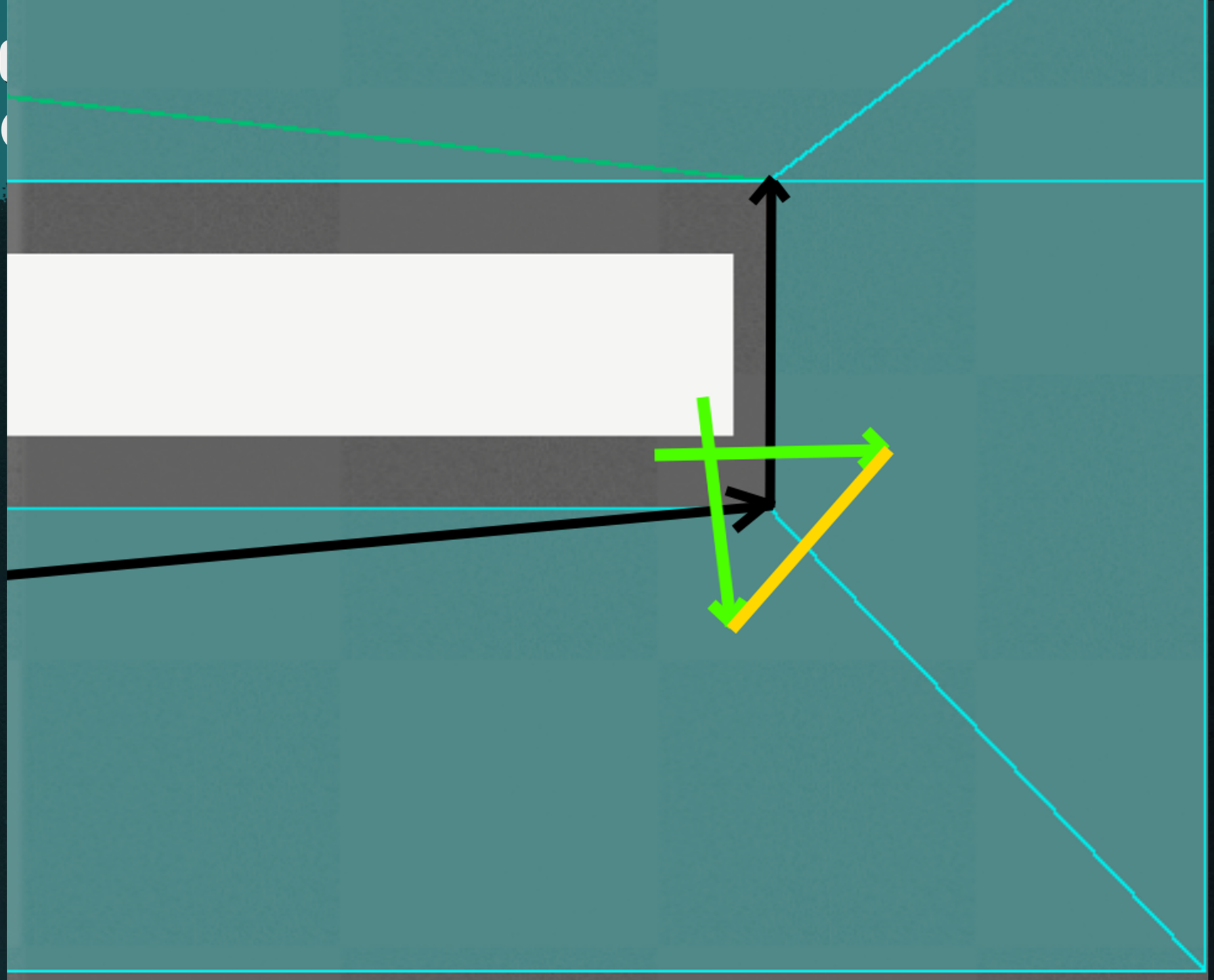
Fit all capsules



Navigati

➢ Path

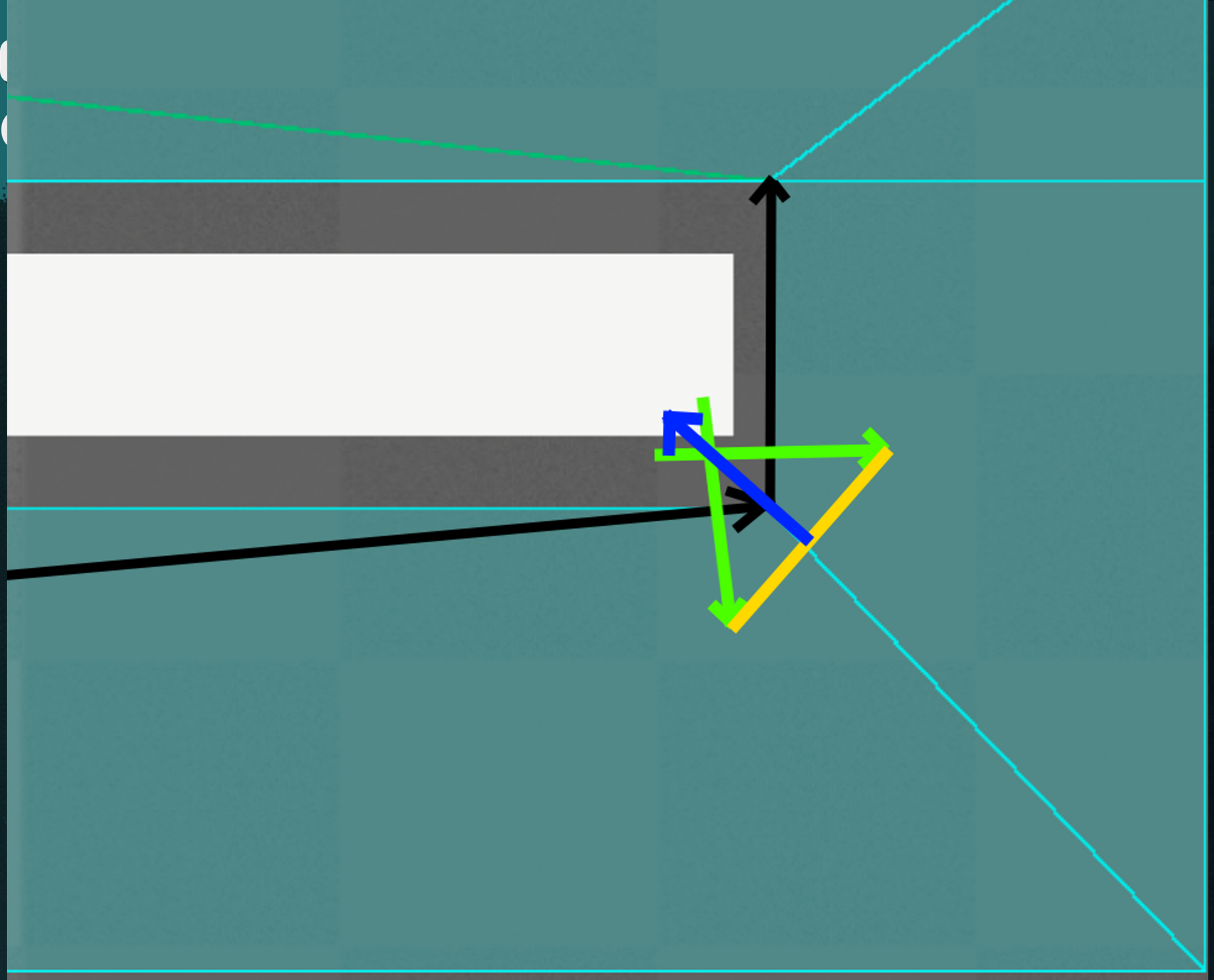
Fit all capsules



Navigati

➢ Path

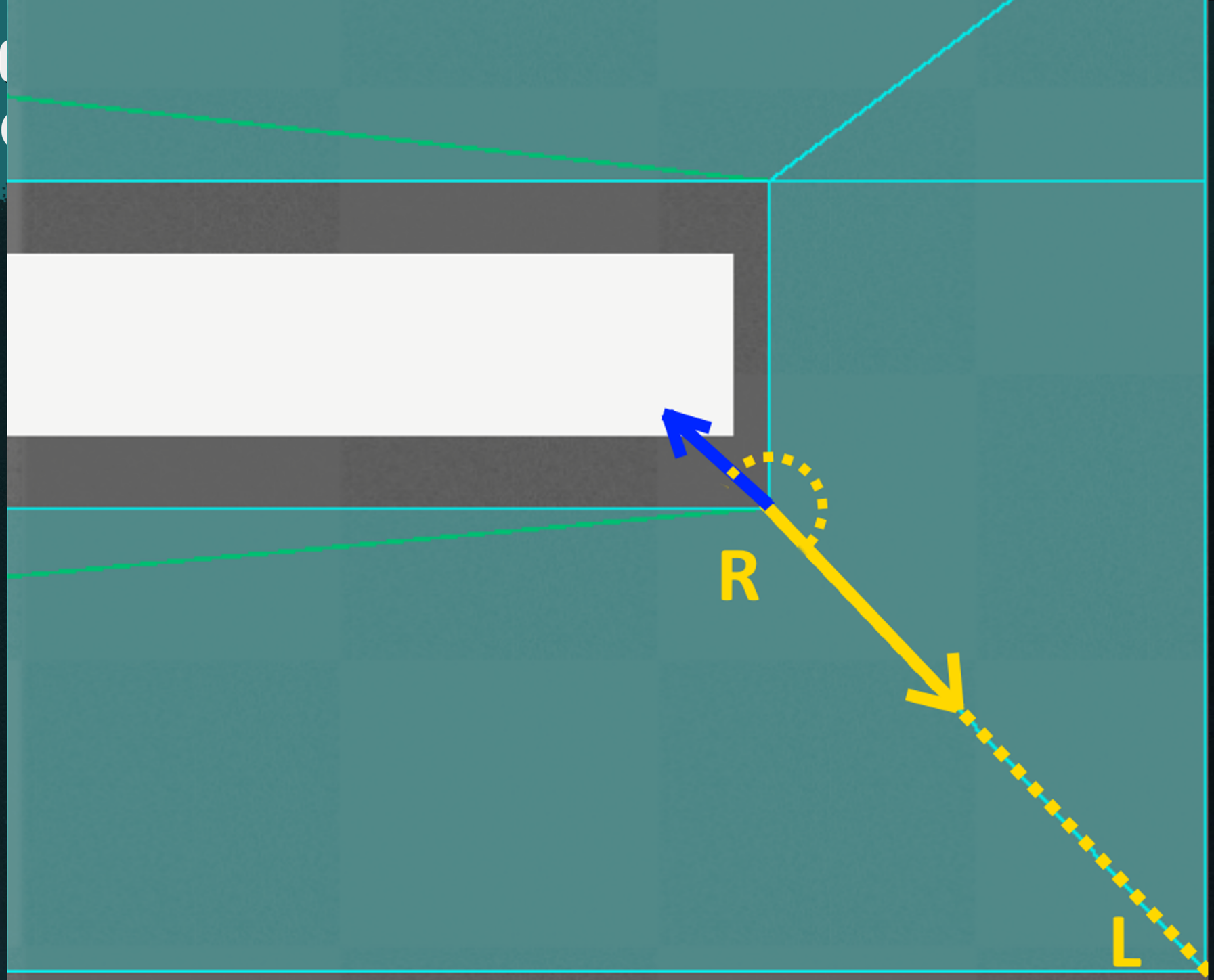
Fit all capsules



Navigati

➢ Path

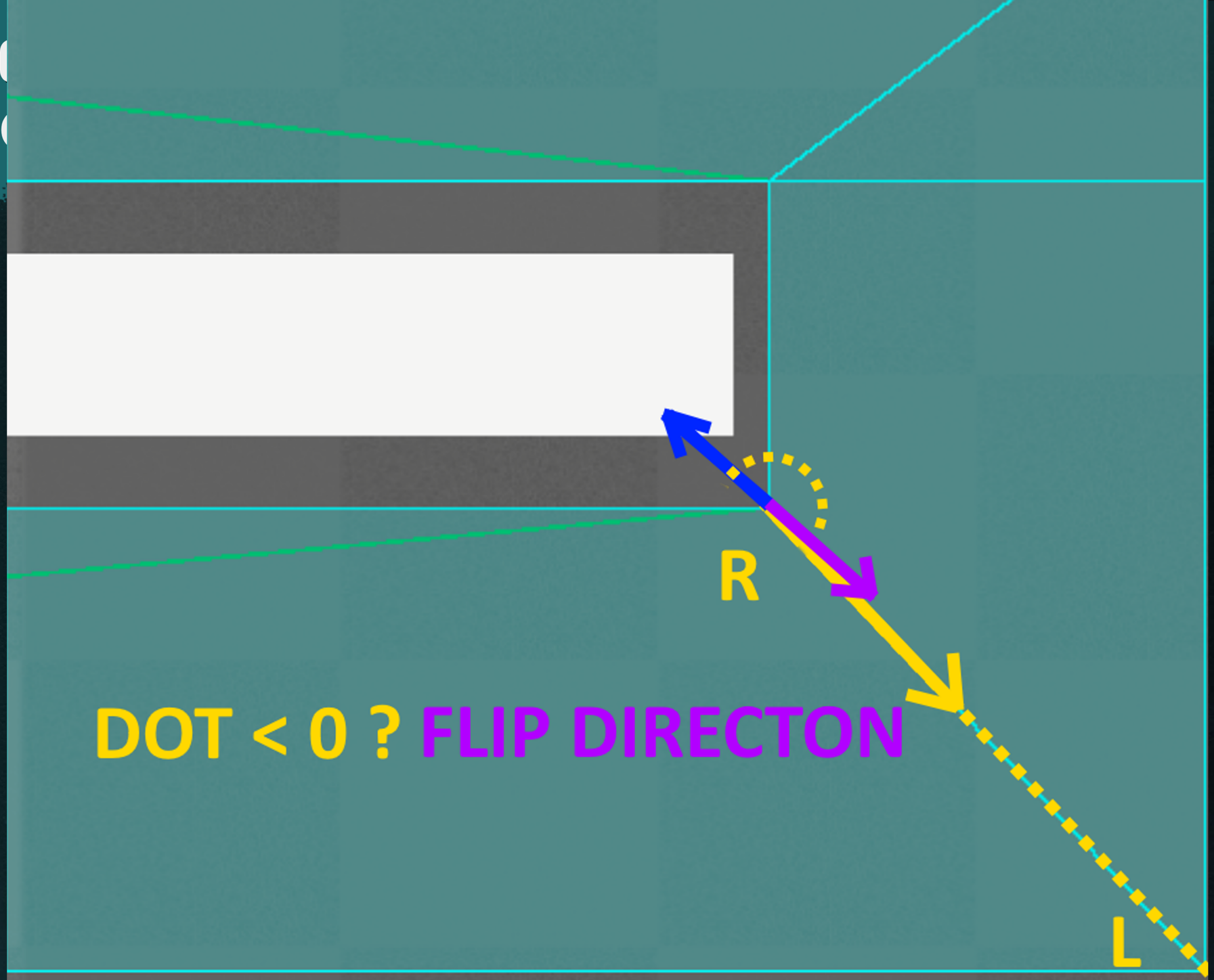
Fit all capsules



Navigati

➢ Path

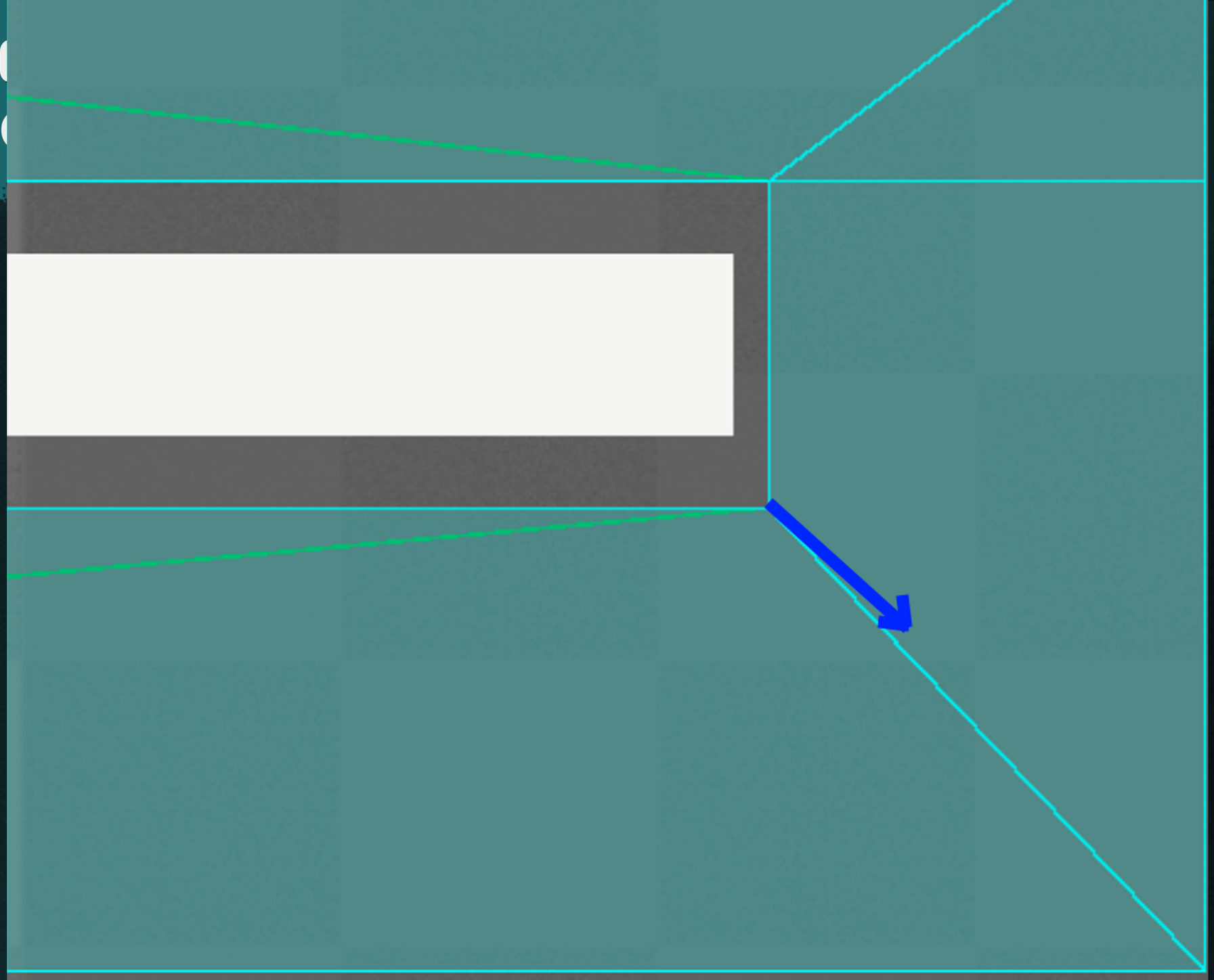
Fit all capsules



Navigati

➢ Path

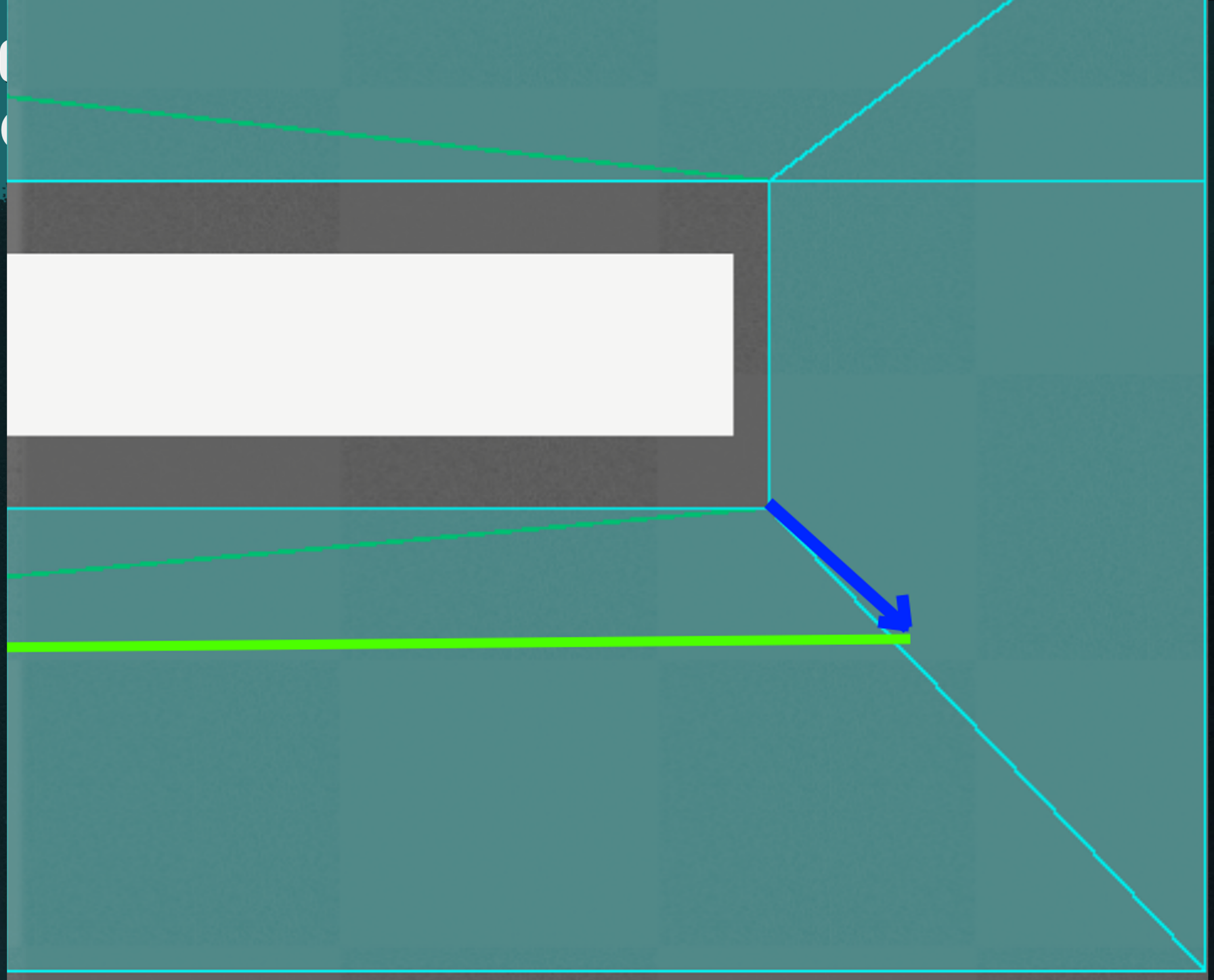
Fit all capsules



Navigati

➢ Path

Fit all capsules



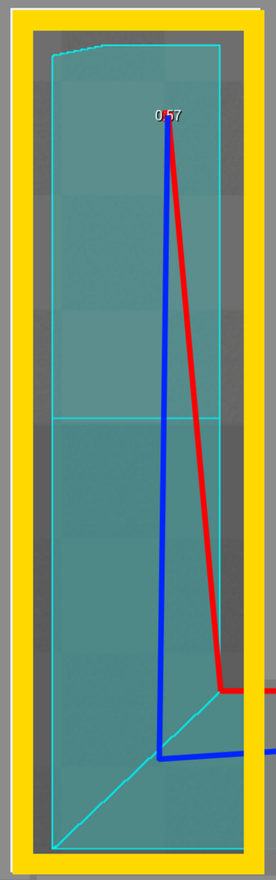
Navigatio

➢ Path C

Fit all capsules

Simple Stupid Funnel

Good Enough Path Offset



Performance

- *Multi-thread*

  0.2s to 0.5s average resolve time
- *Frame Budget*

  Too long? Continue next frame and log
- *Skip Paths*

  Ignore straight paths (two points)
- *Limit active enemies*

  Max 20 active enemies (+ L.O.D.)

Navigation

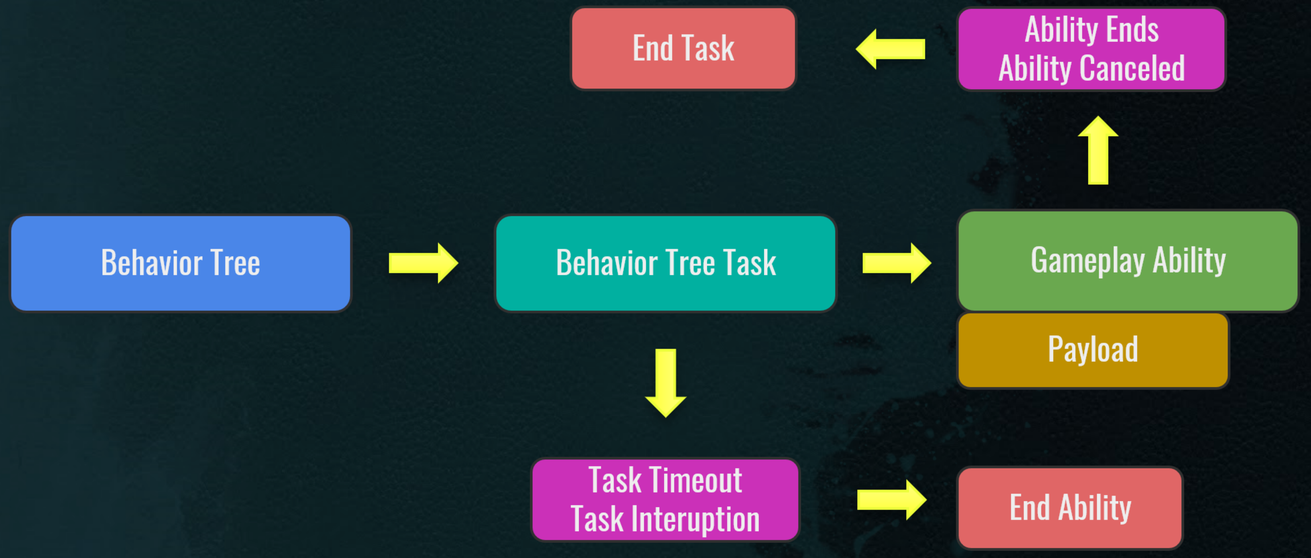
➢ Path Calculation

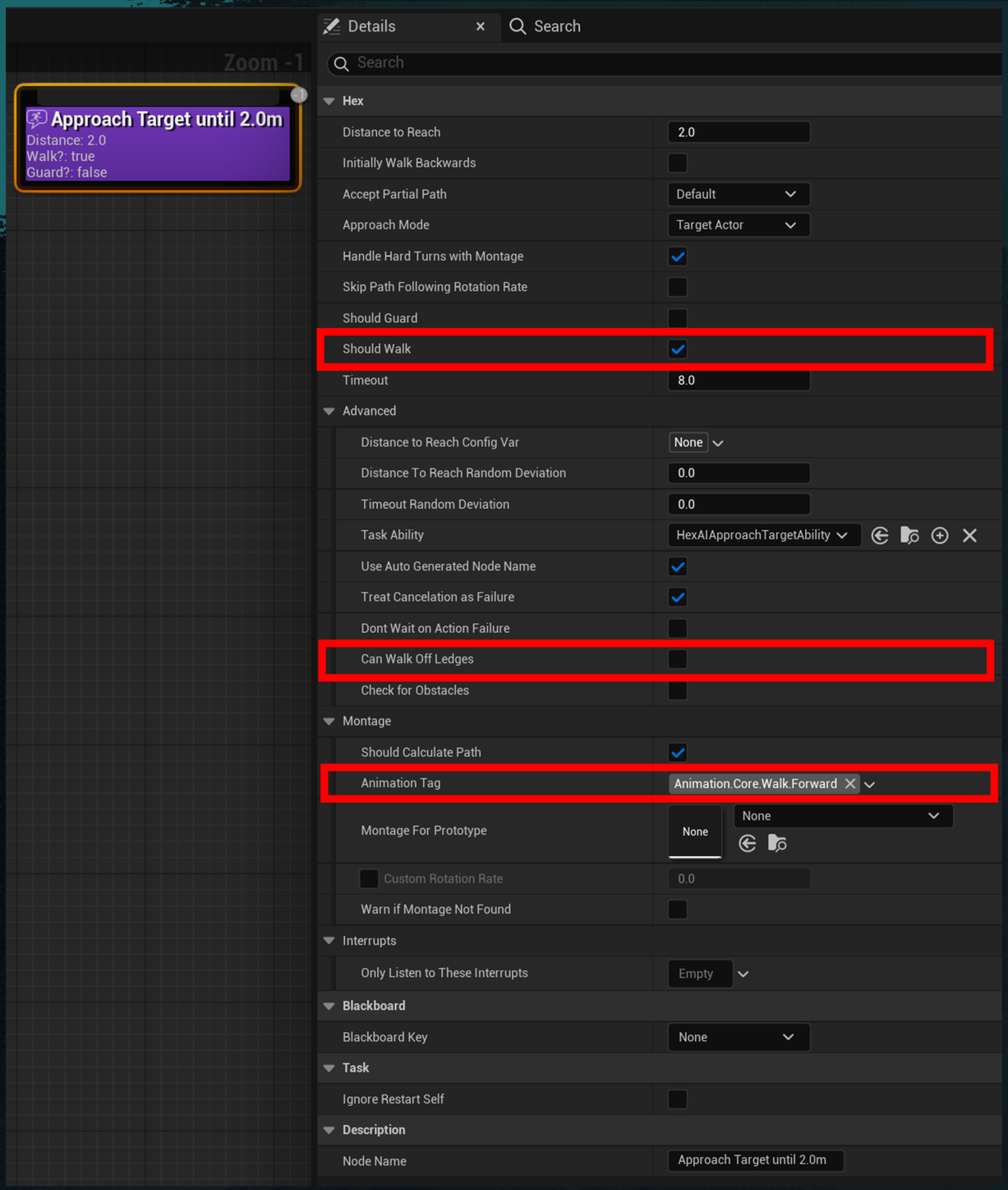
Fit all capsules

DECISION MAKING

Decisions

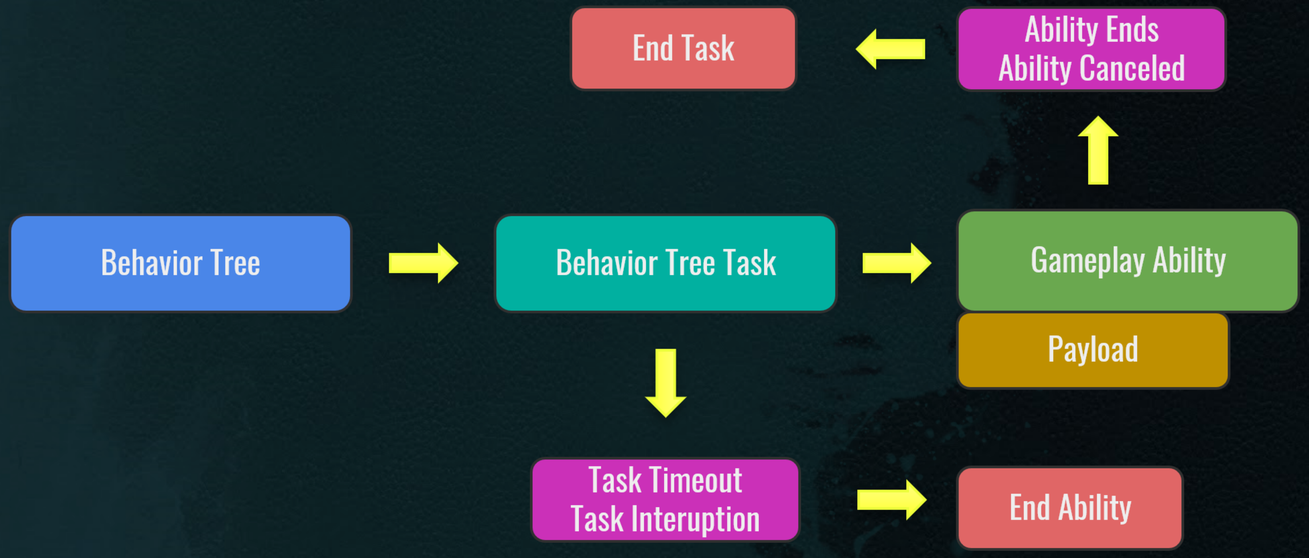
➢Behavior flow

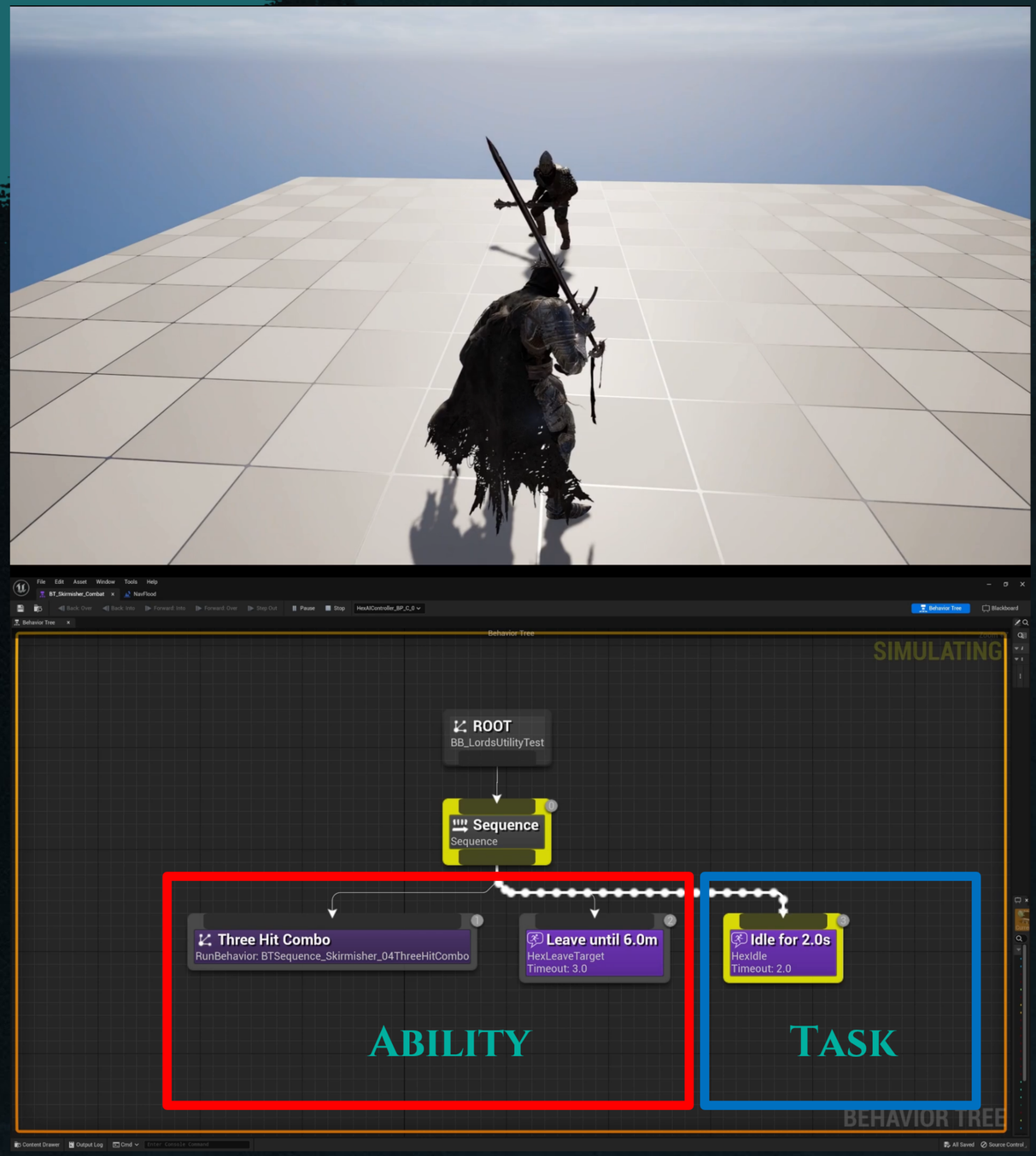




Decisions

➢Behavior flow





Decisions

➢Data Driven Approach

Character Data Asset

- *Equipment*

  Armour, Weapons and Wearables

- *Stats*

  Health, Stamina etc (145 different values)

- *Loot*

  Items drop upon death, probability and quality

- *Gameplay Data*

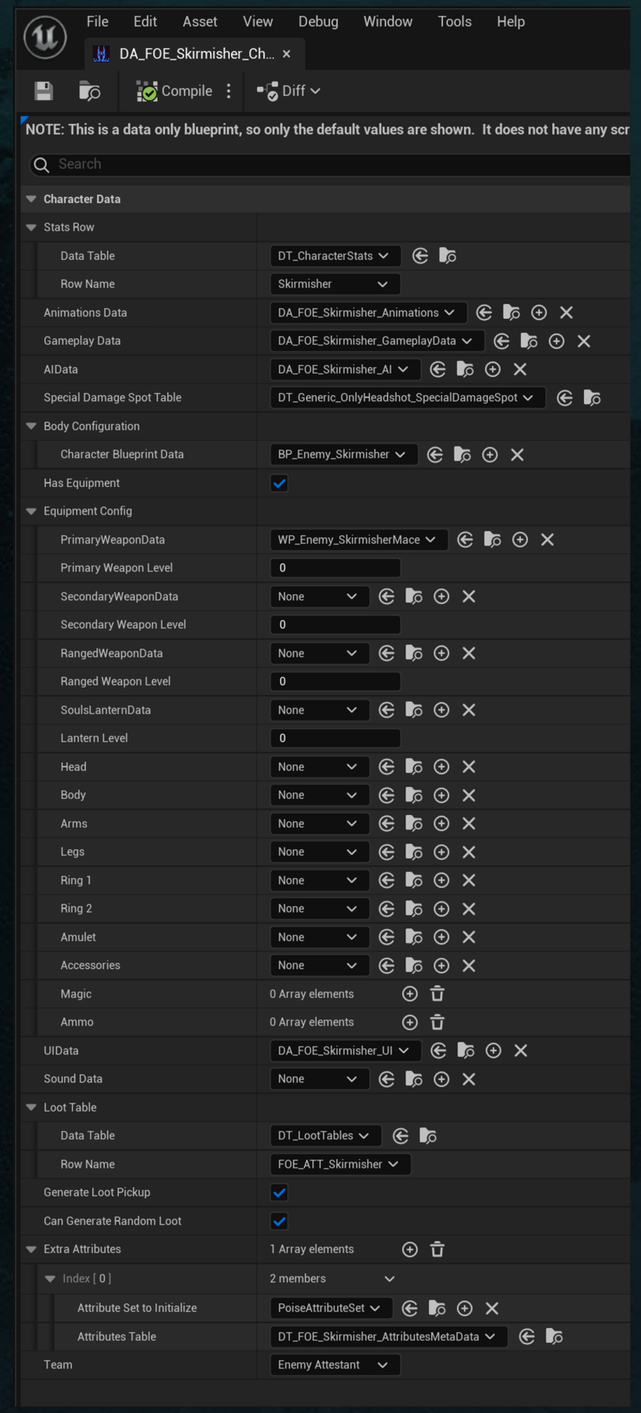
  G.A.S. data (ie: abilities granted), Camera lock-on

- *Animations*

  Montages, Sequences, blending values

- *AI*

  Behaviours, Perception, Navigation, Combat



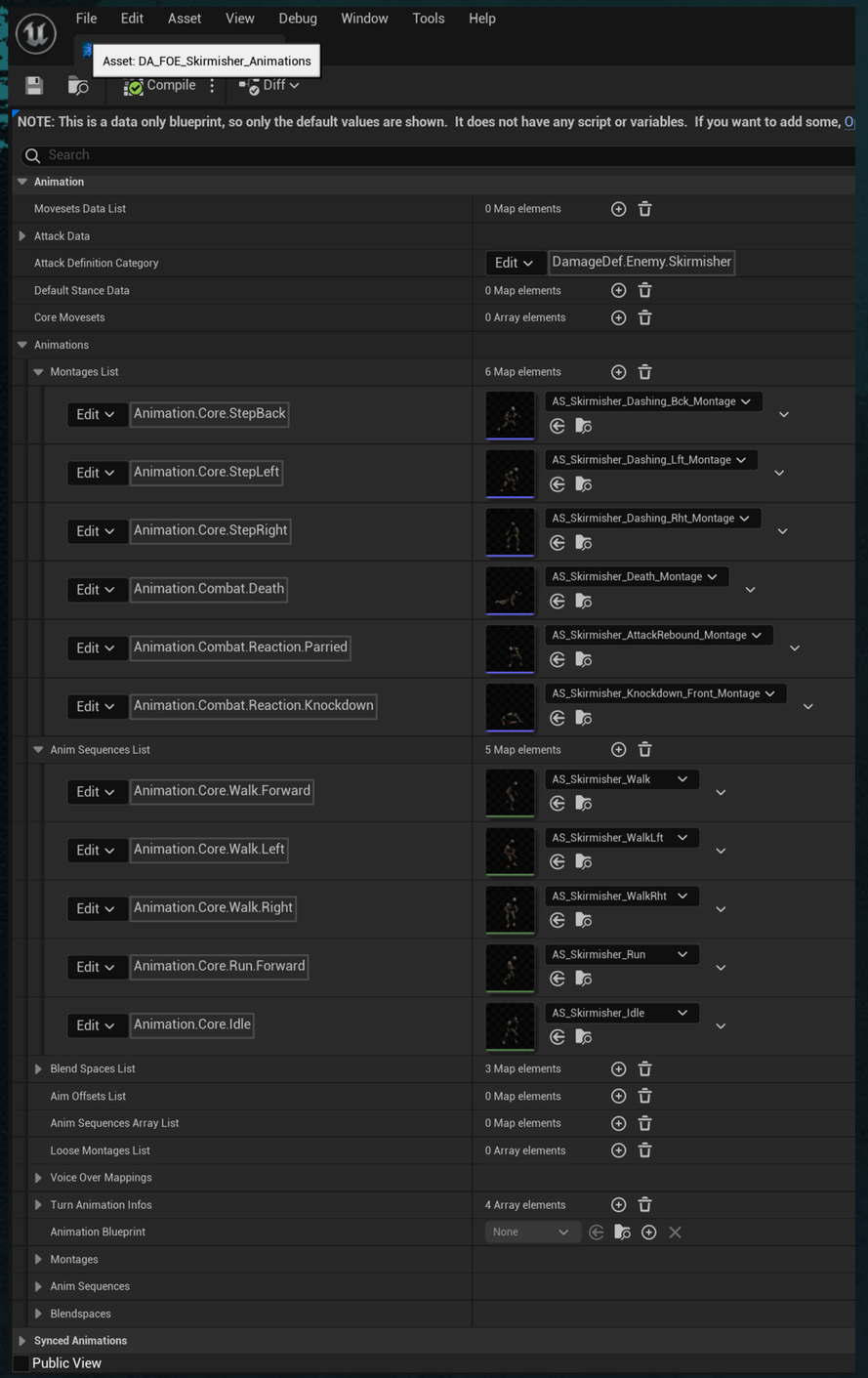
Decisions

➢Data Driven Approach

Character Data Asset

- *Animations*

  Montages, Sequences, blending values



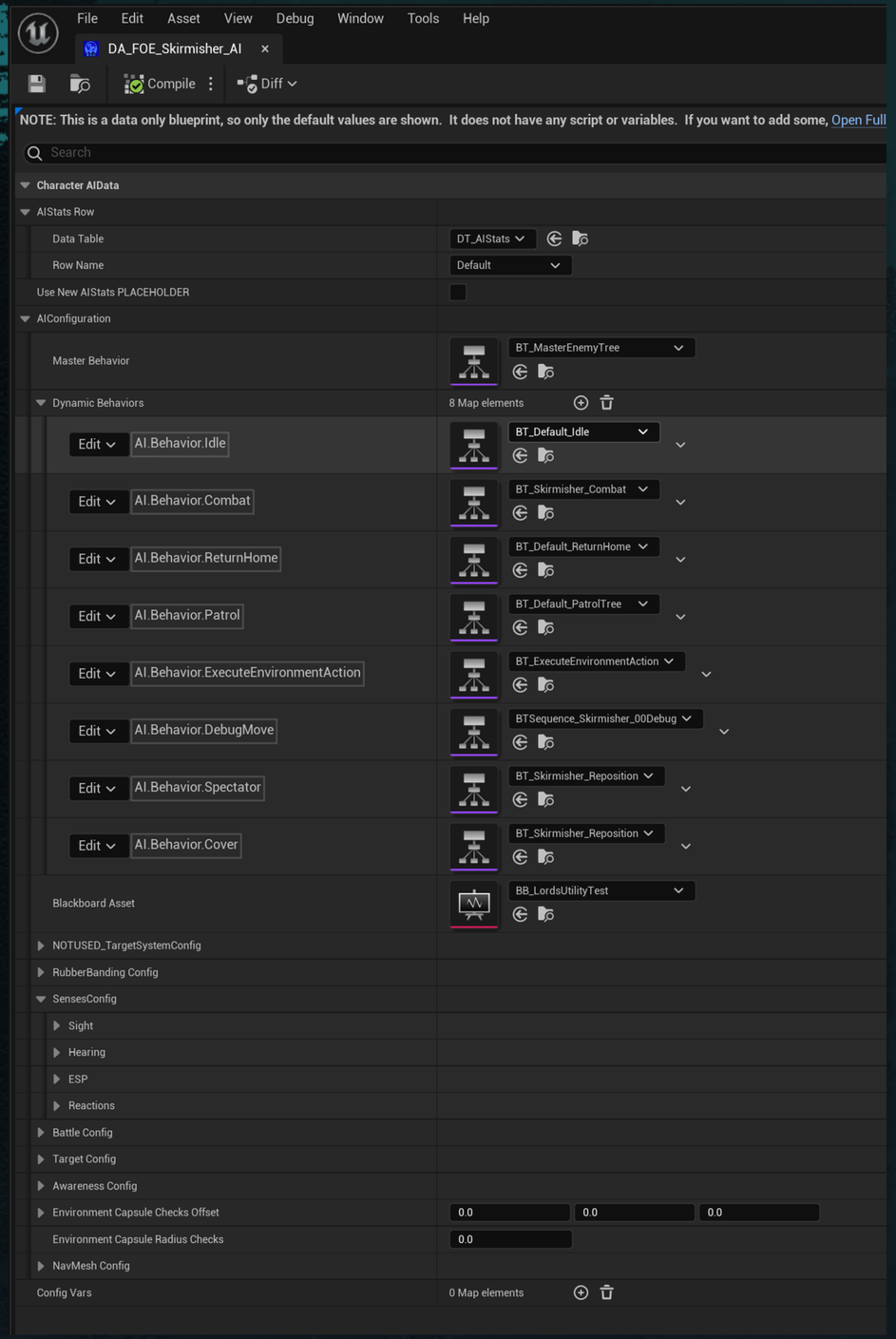
Decisions

➢Data Driven Approach

Character Data Asset

- *AI*

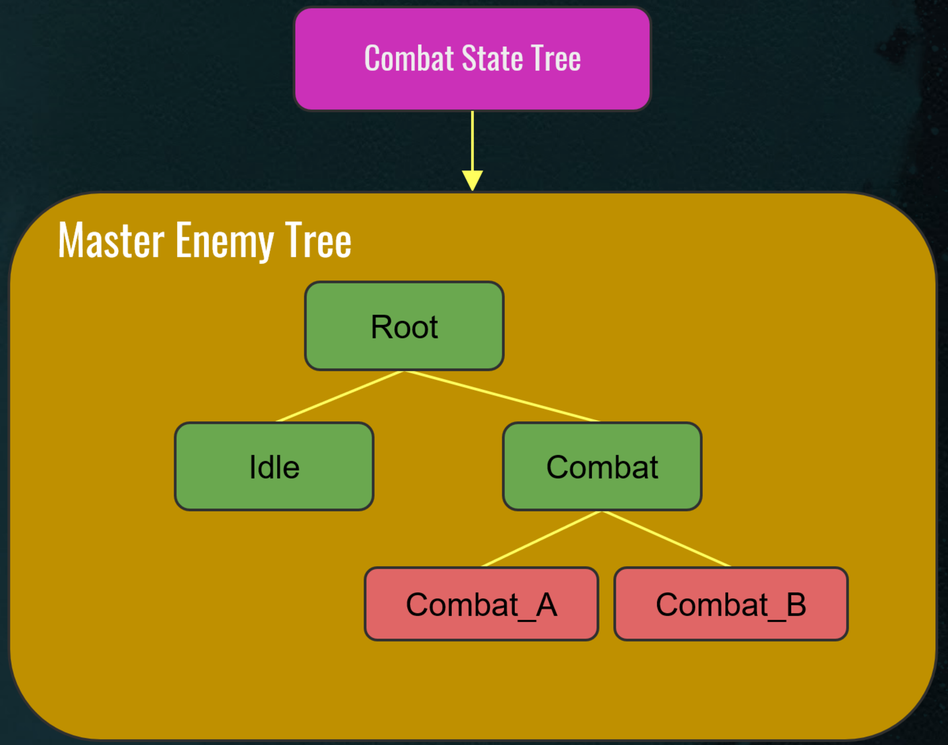
  Behaviours, Perception, Navigation, Combat

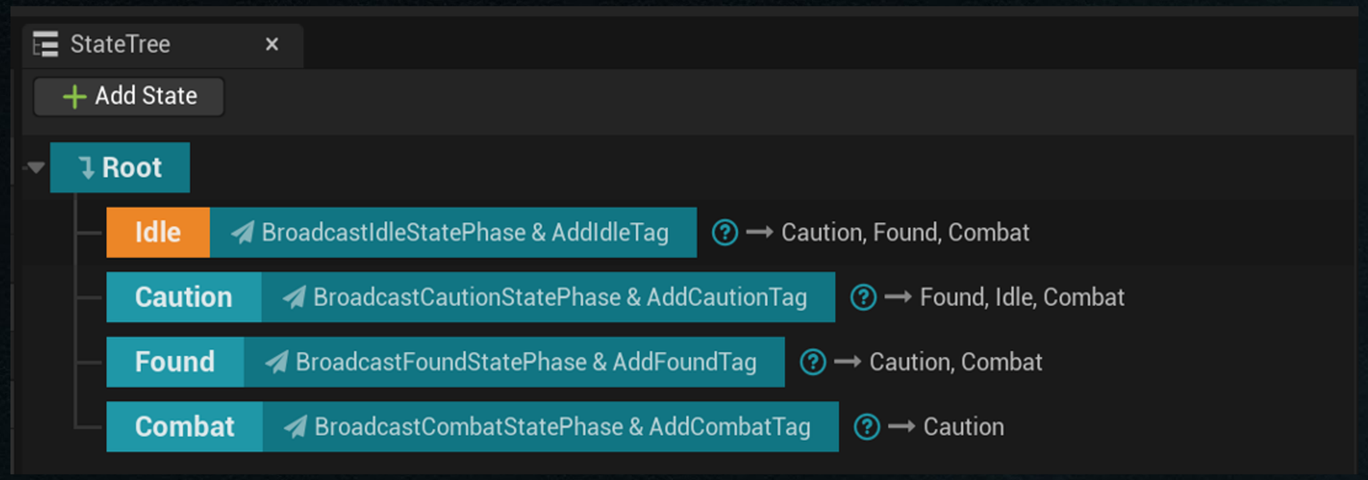


Decisions

➢Combined Approach

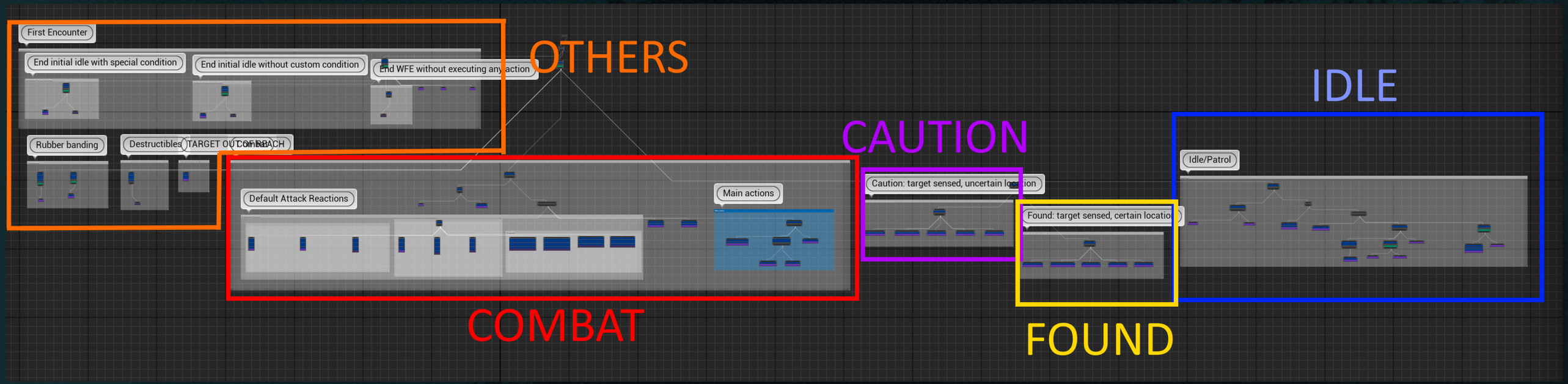
AAIController::OnPossess → Run Master Enemy Tree in DA_AI → Inject all the trees from DA_AI





Decisions

➢Combined Approach



Decisions

➢Reaction System - Overview

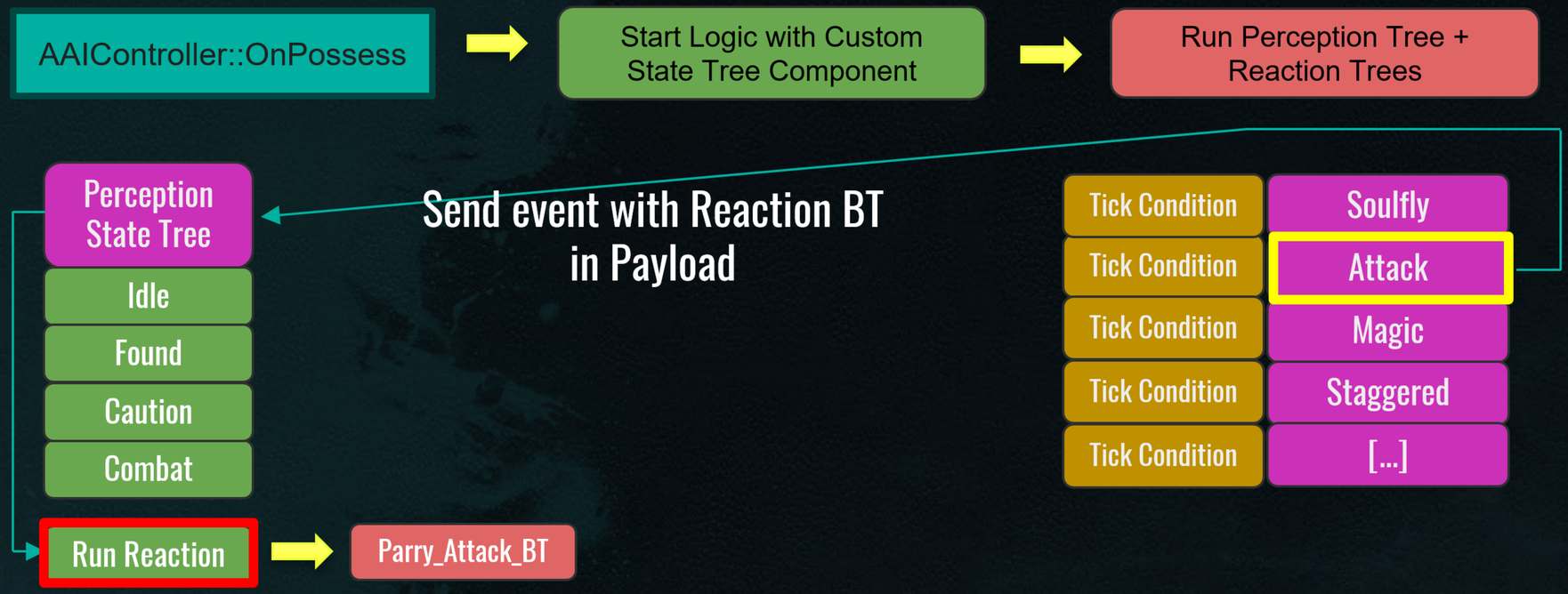
Inspired by "Reactive Behavior Trees" (Sarat Rallabandi and Paul Roberts)

Reaction: anything that is not Idle, Caution, Found, Combat

- Soulfly Parry, Block, Dodge
- Attack Parry, Block, Dodge
- Magic/Projectiles Block, Dodge
- Staggered
- Target Out Of Reach
- Destroy "Desctructibles"
- Rubberbanding
- Wait First Encounter

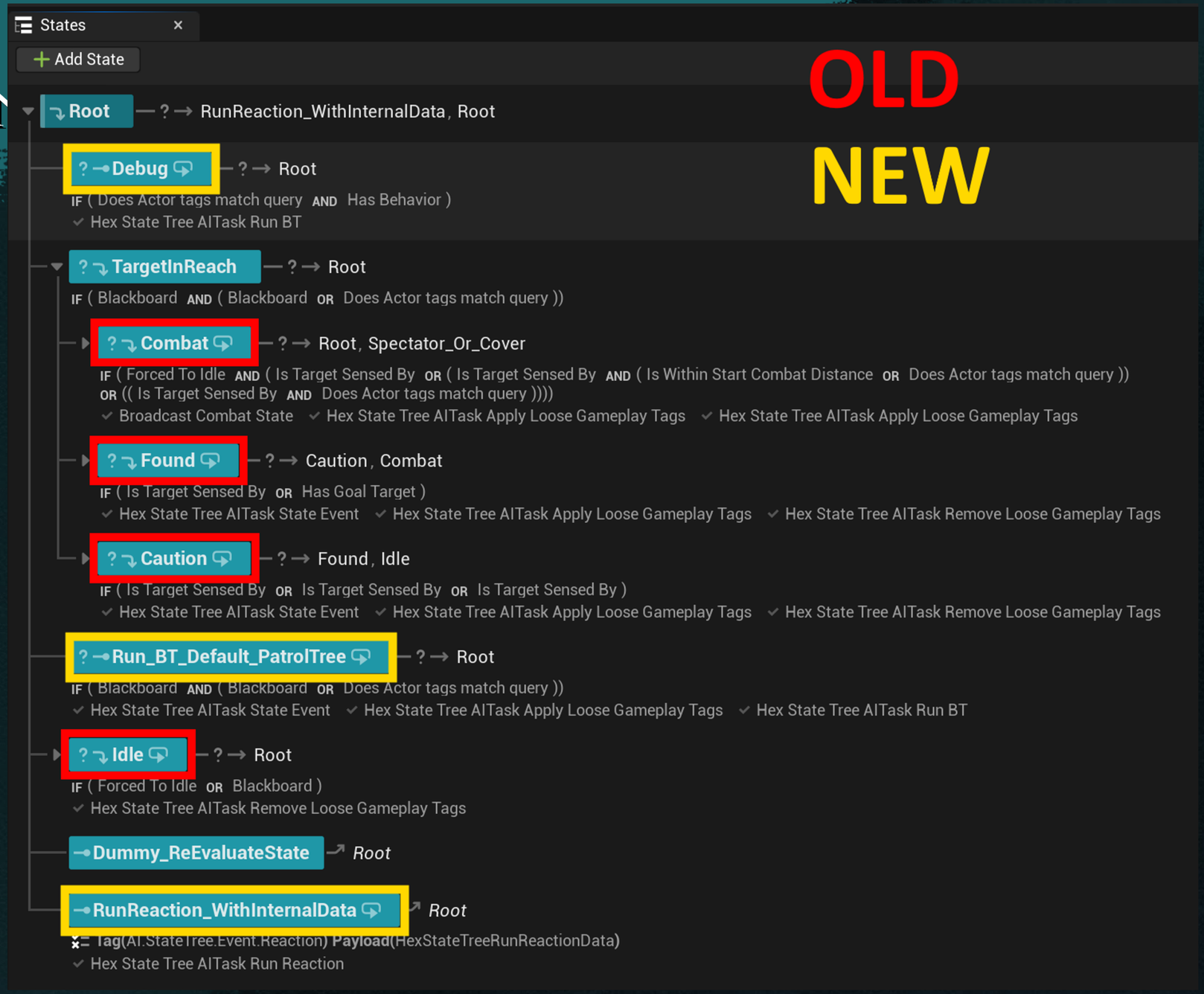
Reaction Tick Condition: conditions to tick a Reaction State Tree

- Gameplay Tags Query
- Gameplay Event



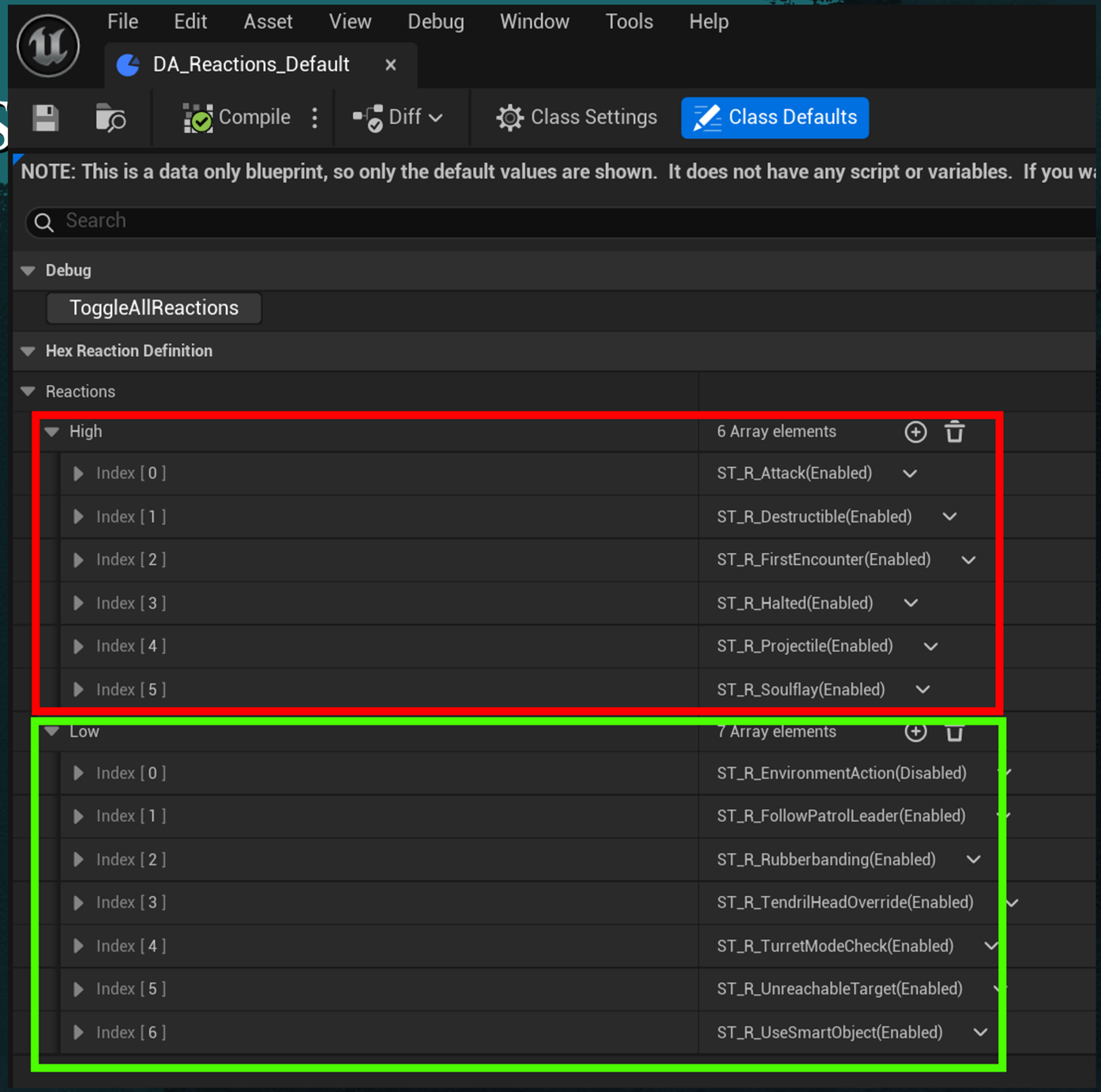
Decisions

➢ REACTION



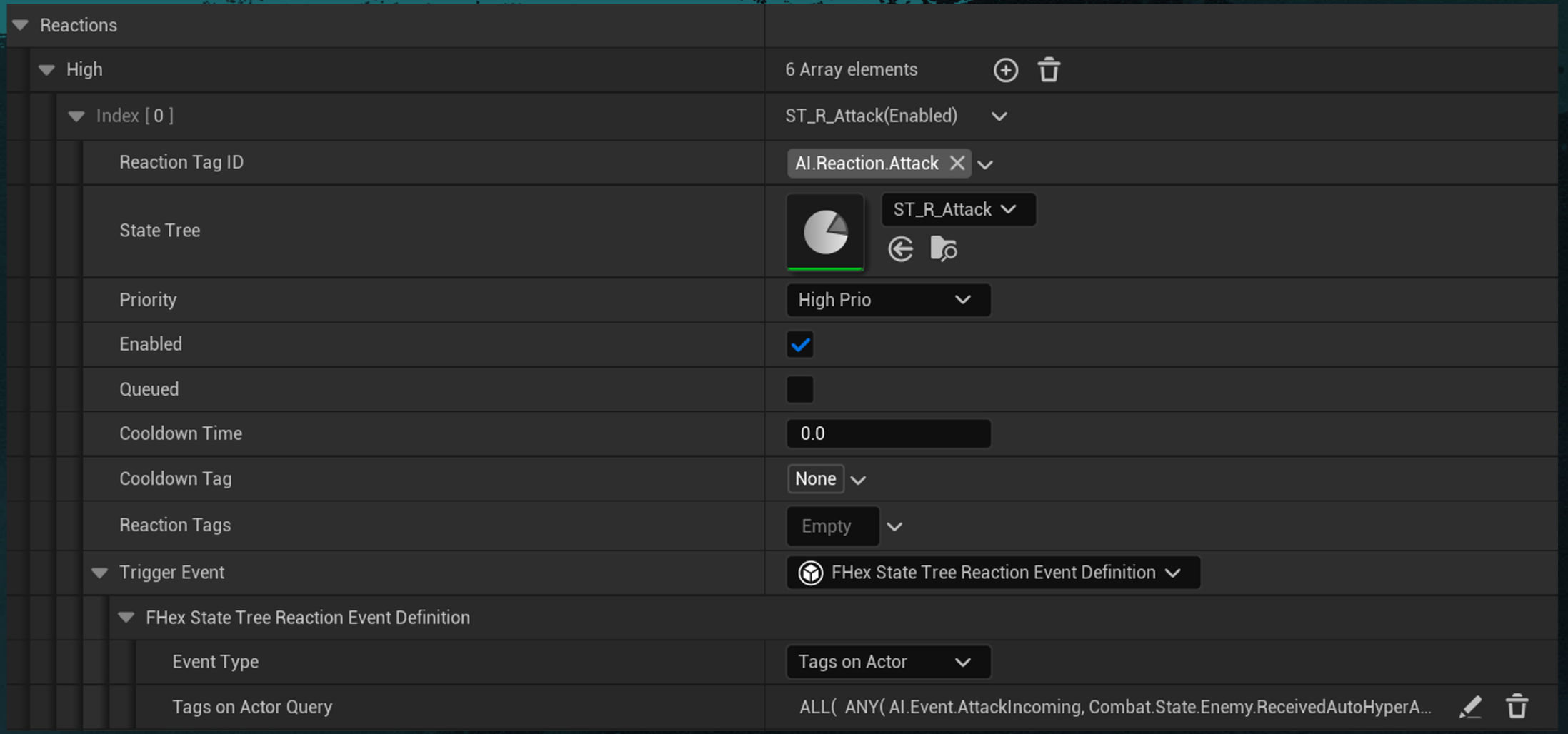
Decisions

➢ Reaction S



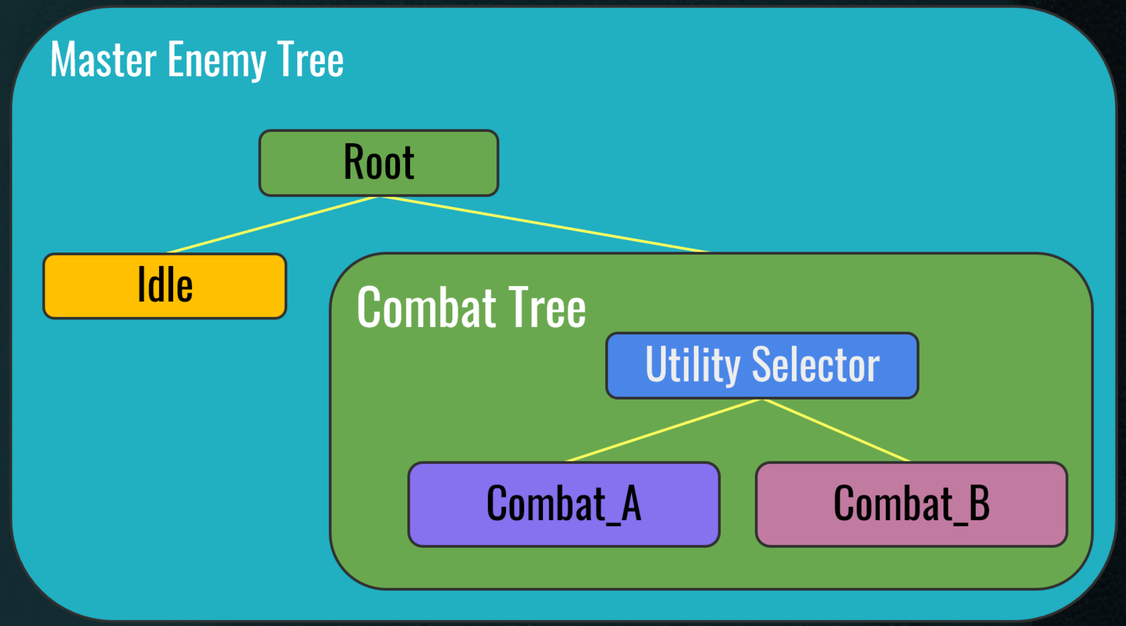
Decisions

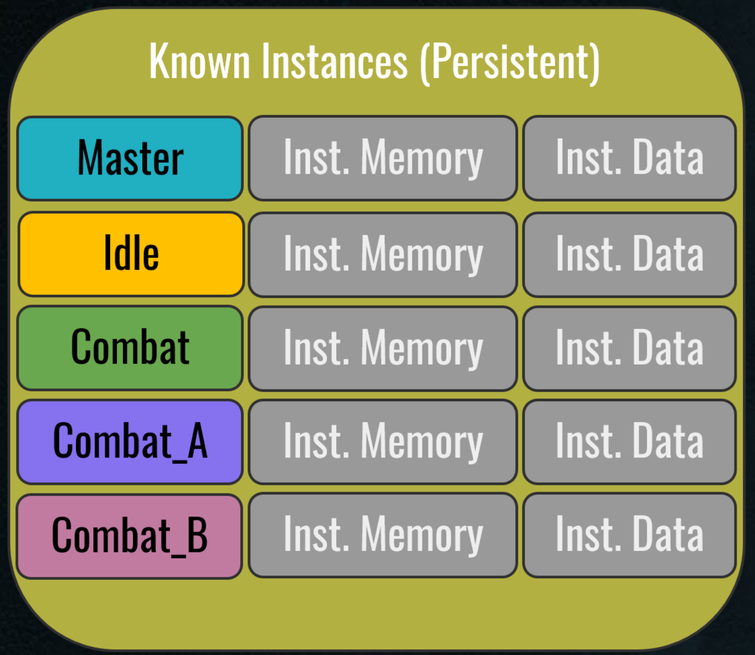
➢Reaction System - Overview



Decisions

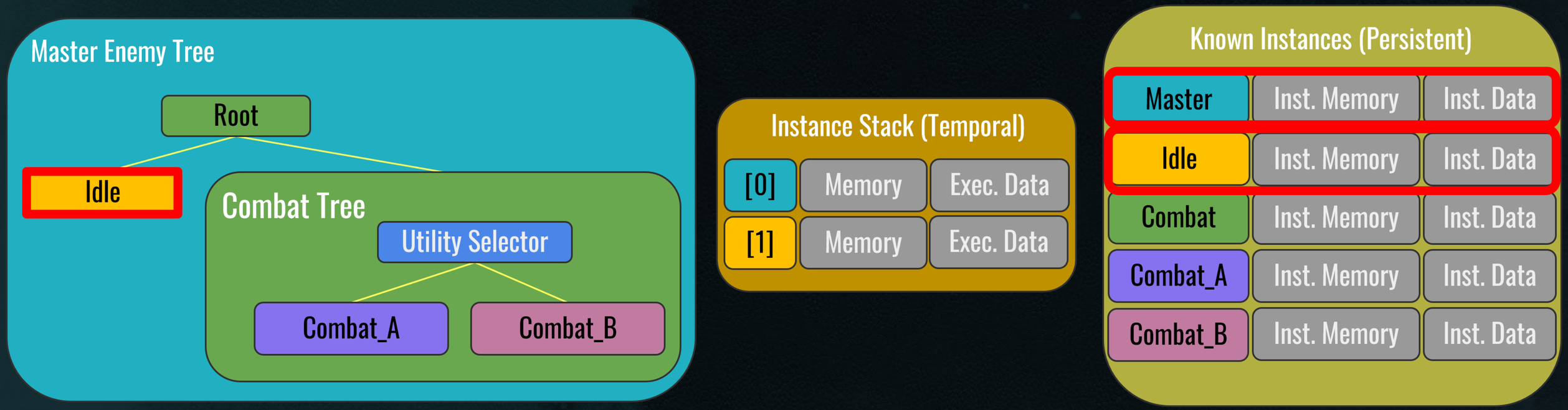
➢Unreal BT Memory Handling





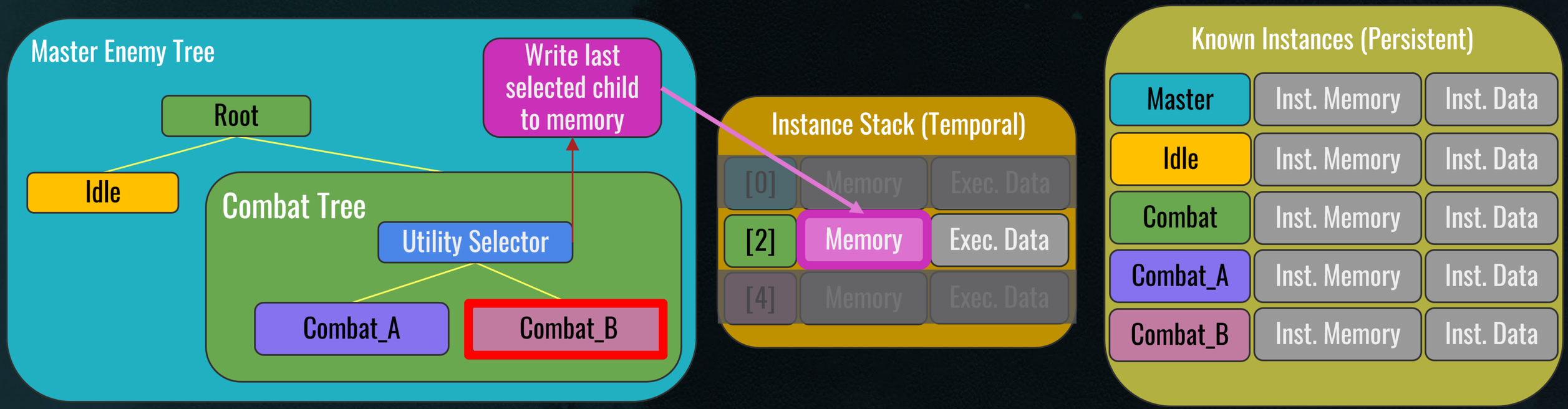
Decisions

➢Unreal BT Memory Handling



Decisions

➢Unreal BT Memory Handling

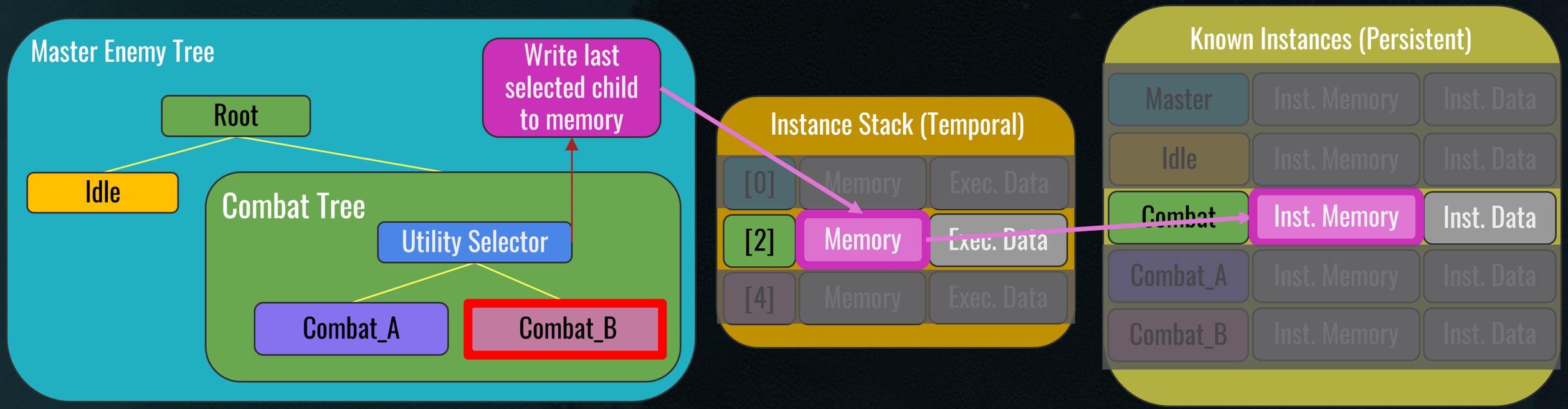


Decisions

➢Unreal BT Memory Handling

Process Execution Request

- **Process Pending Execution:** During Instance Stack Clean up if not destructive
- **Copy Instance Memory To Persistant:** For Rollback safety (after cleaning stack)



Decisions

➢Reaction System Memory Handling

Start

Create Known And Instance Stack

Tick

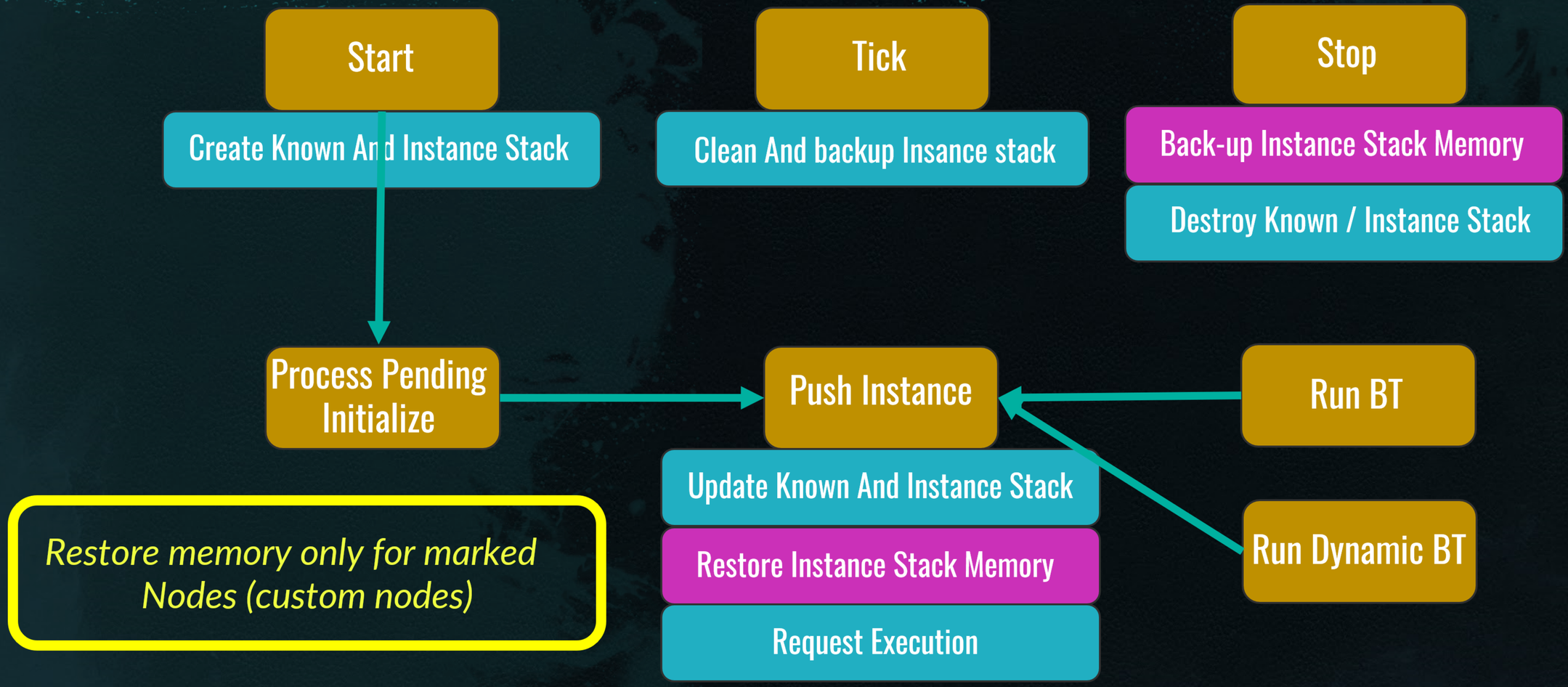
Clean And backup Insance stack

Stop

Destroy Known / Instance Stack

Decisions

➢Reaction System Memory Handling



COMBAT

Combat

➢AGGRO

Combat

➢AGGRO

Combat

➢AGGRO

Combat

➢AGGRO

Combat

➢Combat roles

The end

Thanks!

To Connect

LinkedIn

To Help

Discord Unreal Source

To Remember

Internet Archive

Youtube videos used:

https://www.youtube.com/shorts/-CfJxJwrj14
https://www.youtube.com/shorts/y4gf5FUzaUE
https://www.youtube.com/shorts/gL7nH8SNilQ

Reactive Behavior Trees, by Sarat Rallabandi and Paul Roberts

Game AI Uncovered, Volume one, chapter 7.

https://www.routledge.com/Game-AI-Uncovered-Volume-One/Roberts/p/book/9781032343235